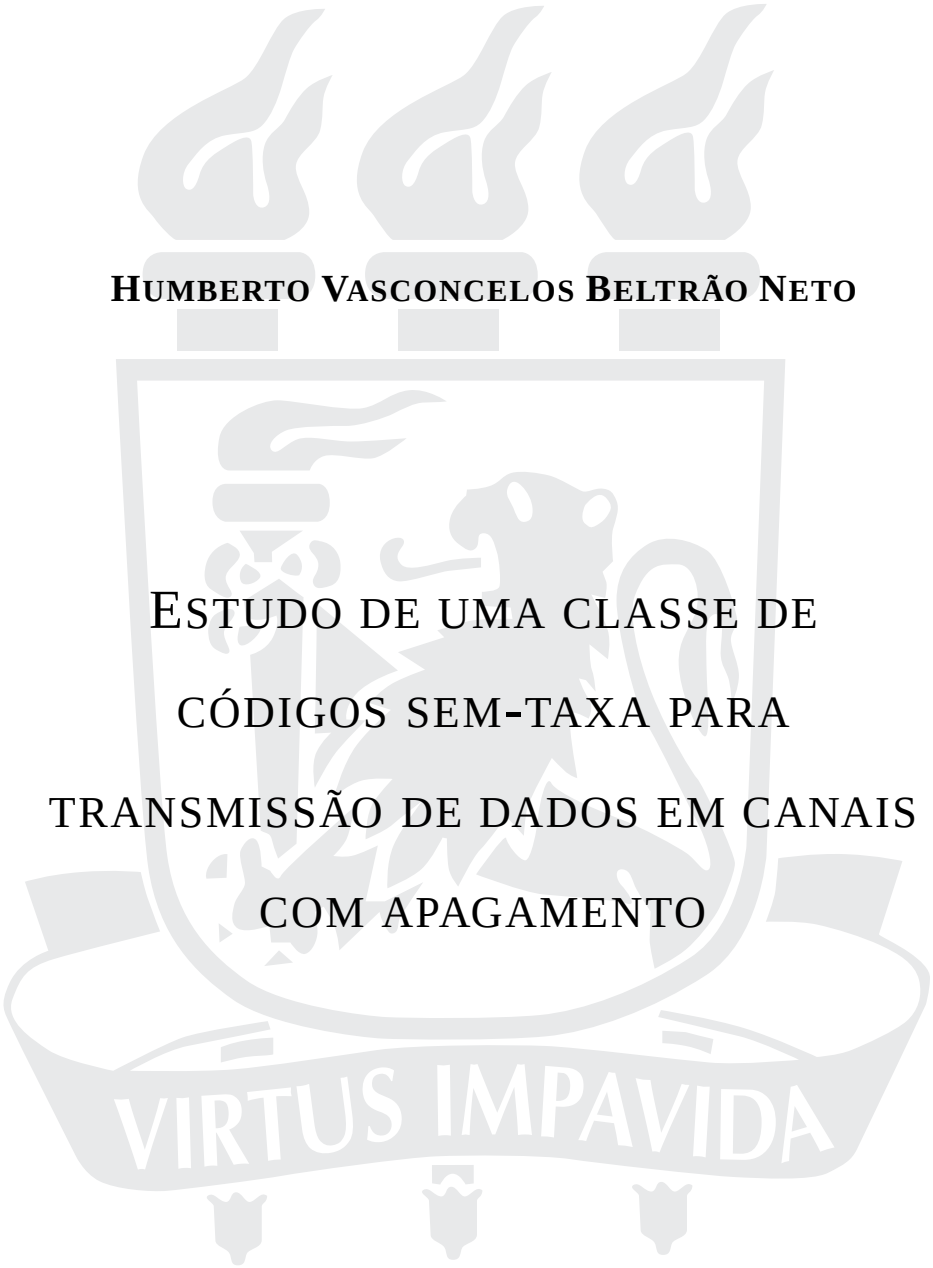
UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE TECNOLOGIA E GEOCIÊNCIAS

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



HUMBERTO VASCONCELOS BELTRÃO NETO

ESTUDO DE UMA CLASSE DE CÓDIGOS SEM-TAXA PARA TRANSMISSÃO DE DADOS EM CANAIS COM APAGAMENTO

RECIFE, MARÇO DE 2007.

HUMBERTO VASCONCELOS BELTRÃO NETO

ESTUDO DE UMA CLASSE DE CÓDIGOS SEM-TAXA PARA TRANSMISSÃO DE DADOS EM CANAIS COM APAGAMENTO

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Pernambuco como parte dos requisitos para obtenção do grau de **Mestre em Engenharia Elétrica**

ORIENTADOR: PROF. VALDEMAR CARDOSO DA ROCHA JÚNIOR, PH.D.

Recife, Março de 2007.

B453e **Beltrão Neto, Humberto Vasconcelos.**
Estudo de uma classe de códigos sem-taxa para transmissão de dados em canais com apagamento. – Recife: O Autor, 2007.
74 folhas. : il. ; fig., tabs.

Dissertação (Mestrado) – Universidade Federal de Pernambuco. CTG. Engenharia elétrica, 2007.

Inclui bibliografia.

1. Engenharia elétrica. 2. Redes de comunicação – Canais com apagamento. 3. Códigos LT – Análise de desempenho. 4. Canais com apagamento – Códigos sem-taxa - Estudo. I. Título.

621.3 CDD (22.ed.) UFPE **BCTG/2007-074**

Universidade Federal de Pernambuco

Pós-Graduação em Engenharia Elétrica

PARECER DA COMISSÃO EXAMINADORA DE DEFESA DE
TESE DE MESTRADO ACADÊMICO DE

HUMBERTO VASCONCELOS BELTRÃO NETO

TÍTULO

"ESTUDO DE UMA CLASSE DE CÓDIGOS SEM TAXA PARA TRANSMISSÃO DE DADOS EM CANAIS COM APAGAMENTO"

A comissão examinadora composta pelos professores: VALDEMAR CARDOSO DA ROCHA JÚNIOR, DES/UFPE, RICARDO MENEZES CAMPELLO DE SOUZA, DES/UFPE e RICHARD DEMO DE SOUZA, CEFET/PR, sob a presidência do primeiro, consideram o candidato **HUMBERTO VASCONCELOS BELTRÃO NETO** **APROVADO**.

Recife, 12 de março de 2007.

JOAQUIM FERREIRA MARTINS FILHO
Coordenador do PPGEE

VALDEMAR CARDOSO DA ROCHA JÚNIOR
Orientador e Membro Titular Interno

RICHARD DEMO DE SOUZA
Membro Titular Externo

RICARDO MENEZES CAMPELLO DE SOUZA
Membro Titular Interno

À minha namorada **Wiviane Souza**,

que com seu amor me acompanhou ao longo

de todo o desenvolvimento desta dissertação,

cedendo horas que por direito seriam suas.

AGRADECIMENTOS

Ao Prof. Valdemar Cardoso da Rocha Júnior, primeiramente por ter acreditado em minha capacidade, depois pela orientação, amizade e atenção sempre dispensadas a mim.

Agradeço também a dois professores que têm sido para mim exemplos a serem seguidos ao longo de toda minha vida, Prof. Ricardo Campello e Prof. Cecílio Pimentel. O conhecimento, dedicação ao ensino e amor à profissão que estes dois homens possuem os tornam mais do que professores, são verdadeiramente mestres.

A meus pais, Antônio e Lúcia, o maior de todos os agradecimentos pela dedicação que ambos tiveram por mim e pela minha formação intelectual, muitas vezes abdicando de seus sonhos para que eu pudesse realizar os meus. Agradecimentos especiais à minha namorada Wiviane Souza, a quem dedico esta dissertação, embora fisicamente eu não pudesse estar sempre ao lado dela, ela esteve comigo durante todo o tempo dedicado a este trabalho.

Aos meus grandes amigos Dyanna Gomes, Fernanda Campello, Leandro Rocha e Marcel Jar, por todos os momentos compartilhados ao longo de toda essa jornada. Lembrando ainda que, se os amigos são a família que nos permitem escolher, esses são os irmãos que escolhi para mim. Agradeço ainda aos colegas de laboratório, especialmente a Márcio Lima, pelo constante companheirismo e conversas ao longo desses dois anos.

E finalmente ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e ao Programa de Pós-Graduação em Engenharia Elétrica, pelo apoio financeiro e a estrutura.

HUMBERTO VASCONCELOS BELTRÃO NETO

Universidade Federal de Pernambuco

12 de Março de 2007

Os que se encantam com a prática sem a ciência são como um timoneiro que entra no navio sem timão nem bússola, nunca tendo certeza acerca do seu destino.

— Leonardo da Vinci

Resumo da Dissertação apresentada à UFPE como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica

ESTUDO DE UMA CLASSE DE CÓDIGOS SEM-TAXA PARA TRANSMISSÃO DE DADOS EM CANAIS COM APAGAMENTO

Humberto Vasconcelos Beltrão Neto

Março/2007

Orientador: Prof. Valdemar Cardoso da Rocha Júnior, Ph.D.
Área de Concentração: Comunicações
Palavras-chaves: códigos LT, códigos sem-taxa, fontes digitais, códigos corretores de erros, canais com apagamento.
Número de páginas: 74

Canais com apagamento têm sido extensivamente aplicados como modelos apropriados para transmissão de dados em redes de comunicação baseadas em pacotes, as quais encontram na Internet seu mais notável exemplo. O estudo de códigos capazes de lidar com apagamentos, garantindo confiabilidade e eficiência na utilização de tais canais, tem sido objeto de muita pesquisa em teoria das comunicações, estudos estes que originaram conceitos como fontes digitais e códigos sem-taxa. Neste contexto surgem os códigos LT. Como uma implementação prática de códigos sem-taxa, esses códigos aproximam-se do conceito de fontes digitais, sendo capazes de recuperar, com probabilidade $(1-\delta)$, um conjunto de k símbolos de entrada a partir de quaisquer $k + O(\sqrt{k}.\ln^2(k/\delta))$ símbolos codificados, necessitando para isso de uma média de $O(k.\ln(k/\delta))$ operações. O estudo dessa recém-criada classe de códigos, incluindo análise de desempenho, técnicas de projeto e possíveis aplicações, constitui um dos objetivos desta dissertação. Um outro objetivo é a descrição de um método para geração de cabeçalhos para códigos LT, o qual necessita de poucos *bits* para sua representação e permite uma decodificação sem necessidade de sincronismo entre codificador e decodificador.

Abstract of Dissertation presented to UFPE as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering

STUDY OF A CLASS OF RATELESS CODES FOR DATA TRANSMISSION OVER ERASURE CHANNELS

Humberto Vasconcelos Beltrão Neto

March/2007

Supervisor: Prof. Valdemar Cardoso da Rocha Júnior, Ph.D.
Area of Concentration: Communications
Keywords: LT codes, rateless codes, digital fountains, FEC, erasure channels.
Number of pages: 74

Erasure channels have been extensively used as appropriate models for data transmission over packet based communication networks, which have the Internet as its most notable example. The research on codes capable of dealing with erasures, guaranteeing reliability and efficiency on the utilization of such channels, has been the subject of much research on communications theory. This research originated concepts such as digital fountains and rateless codes. In this context, LT codes were created. As a practical implementation of rateless codes, these codes are good approximations of the digital fountain concept, being capable of recovering, with probability at least $(1-\delta)$, a set of k input symbols from any set of $k+O(\sqrt{k}.\ln^2(k/\delta))$ output symbols, with an average of $O(k.\ln(k/\delta))$ operations for successful decoding. The description of this novel class of codes, design techniques and applications, is one of the objectives of this dissertation. A second objective is the description of a method for generating short headers for LT codes, which makes possible asynchronous decoding.

LISTA DE FIGURAS

LISTA DE TABELAS

SÍMBOLOS IMPORTANTES

k	Número de símbolos de entrada
n	Número de símbolos codificados
δ	Limite superior da probabilidade de falha na decodificação
β	Razão entre o número de símbolos de saída e o número de símbolos de entrada
γ	Valor médio do grau dos nós de verificação
$1+\epsilon$	Razão entre o número de símbolos de saída e o número de símbolos de entrada
$\Omega(x)$	Distribuição de probabilidade do grau dos símbolos codificados
c	Parâmetro de ajuste da distribuição Sóliton Robusta
R	Valor esperado do número nós de verificação de grau um (utilizando a distribuição Sóliton Robusta)
$\alpha(x)$	Distribuição de probabilidade do número de filhos dos nós “OU”
$\nu(x)$	Distribuição de probabilidade do número de filhos dos nós “E”
$\rho(x)$	Distribuição Sóliton Ideal
$\mu(x)$	Distribuição Sóliton Robusta
$h_l(d)$	Número de símbolos de grau d na l-ésima iteração
ε	Valor esperado do número total de ramos em um grafo

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

Se distância significasse perda, não teríamos
aqui a luz do sol nem a das estrelas.

— Desconhecido

MUitos problemas concernentes a sistemas de comunicação atuais envolvem usuários trocando informação a partir de um canal que pode corromper ou perder parte dessa informação ao longo da transmissão. Um exemplo importante deste cenário é a comunicação realizada por meio de redes de computadores, dentre as quais o maior e talvez mais notável exemplo é a Internet. A troca de informação em tais redes é feita através de pacotes, onde cada pacote pode conter toda ou parte da mensagem original [1]. Ao longo da transmissão, pacotes podem ser corrompidos devido ao ruído ou até mesmo perdidos devido a mecanismos de controle de tráfego, por exemplo.

A abordagem mais utilizada para lidar com a perda de pacotes nestes sistemas é requerer a retransmissão dos pacotes corrompidos ou perdidos, através de um procedimento conhecido como *automatic repeat request* (ARQ) [2]. Sistemas que fazem uso dessa abordagem empregam protocolos, como por exemplo o *transport control protocol* (TCP) [1], nos quais cada um dos receptores envolvidos confirmam a recepção dos pacotes, e, caso não tenham recebido algum deles ou caso algum dos pacotes recebidos esteja corrompido, pedem retransmissão dos mesmos. Existe, no entanto, uma vasta gama de situações em que tal estratégia revela-se altamente dispendiosa, requerendo um grande número de retransmissões para que os receptores sejam capazes de receber a informação original por completo. Um exemplo de situação em que protocolos de retransmissão não são adequados é o caso de transmissões do tipo *multicast*, na qual um transmissor envia mensagens a um determinado grupo de receptores. Uma alternativa para lidar com o problema da retransmissão é empregar códigos

corretores de erros [3], mais precisamente códigos para canais com apagamentos [4, 5].

A estratégia dos códigos corretores de erros consiste em usar um esquema de codificação, o qual é aplicado a cada mensagem a ser transmitida que só então é enviada através de um canal de comunicação. Desta forma, pode-se enviar informação eficientemente mesmo na presença de ruído e perda de pacotes, sendo o receptor capaz de recuperar a mensagem original a partir de um conjunto de dados o qual possua, pelo menos, o mesmo número de pacotes da informação transmitida. É fácil ver que o uso de tais esquemas de codificação elimina a necessidade de retransmissões. O receptor só precisa esperar até receber uma quantidade suficiente de pacotes codificados. É neste cenário que surgem os códigos Luby Transform (LT) [6].

Os códigos LT formam uma classe de códigos criados para transmissão de dados através de canais os quais podem ser modelados como canais com apagamento. Assim, os códigos LT podem ser utilizados em transmissões nas quais pacotes são perdidos ou corrompidos, visto que estes podem ser interpretados pelo decodificador como apagamentos [2, 7]. Códigos LT são capazes de recuperar, com probabilidade no mínimo $(1 - \delta)$, um conjunto de k símbolos de entrada a partir de quaisquer $k + O(\sqrt{k}.\ln^2(k/\delta))$ símbolos codificados, necessitando para isso de uma média de $O(k.\ln(k/\delta))$ operações "ou-exclusivo".

Códigos tradicionais para canais com apagamento são tipicamente códigos de bloco com taxa fixa, i.e., k símbolos de entrada são usados para gerar $n - k$ símbolos redundantes para um total de n símbolos codificados, sendo a taxa do código dada por k/n [8]. Tendo em vista que, para códigos LT, o número de símbolos codificados que podem ser gerados a partir dos dados originais é potencialmente ilimitado, os mesmos são classificados como sem-taxa (*rateless codes*) [4, 5, 6, 9]. Dentre os códigos sem-taxa, podemos citar ainda os códigos Raptor [10], os códigos em tempo real [11] e os códigos Slepian-Wolf sem-taxa [12].

O objetivo dessa dissertação é investigar características e aplicações de códigos LT. Para tanto, são expostas algumas ferramentas para análise da probabilidade de falha na decodificação, assim como ferramentas para a construção de distribuições de graus capazes de gerar bons códigos LT. Como contribuição, é proposto um esquema de geração de cabeçalhos de pequeno comprimento (número de *bits*) para códigos LT, os quais tornam o processo de decodificação completamente assíncrono, reduzindo assim a complexidade do sistema. A função de tais cabeçalhos consiste em informar ao decodificador, quantos e quais são os vizinhos de cada símbolo codificado, uma informação essencial ao processo de decodificação.

O esquema proposto mostra-se eficiente em dois aspectos: 1) necessita de um pequeno número

de *bits* para sua construção; e 2) permite uma comunicação sem a necessidade de sincronismo entre o codificador e o decodificador. A primeira característica faz com que se minimize a quantidade de informação auxiliar, maximizando a quantidade de informação "útil" em cada símbolo codificado. Conforme será visto no Capítulo 2, a razão entre o número de *bits* necessários à representação da informação auxiliar e o número de *bits* contendo informação útil depende apenas do número de *bits* dos símbolos de entrada, podendo ser esta ajustada de acordo com as necessidades do sistema. A segunda característica elimina um requisito que por vezes acrescenta complexidade aos sistemas de comunicação: a necessidade de sincronismo. O casamento dessas duas características gera um esquema de representação de cabeçalhos de fácil implementação e baixa complexidade conforme será exposto na Seção 2.3.

1.1 Comunicação em canais ruidosos

O problema fundamental em teoria das comunicações consiste em reproduzir em um ponto, exata ou aproximadamente uma mensagem selecionada em outro ponto. Sempre que uma fonte deseja enviar uma mensagem para um receptor, ela o faz através de um canal de comunicação. No entanto, canais de comunicação reais são sempre afetados por ruído, o qual é constituído por sinais aleatórios advindos de fontes as quais podem ser internas ou externas ao sistema de comunicação utilizado. Fontes externas incluem interferências causadas por sinais sendo transmitidos em canais próximos, lâmpadas fluorescentes, tempestades elétricas e raios. Já as fontes internas de ruído incluem a movimentação de elétrons em condutores, a difusão e recombinação de portadores eletricamente carregados em aparelhos eletrônicos e emissões aleatórias. O ruído é sem dúvida um dos fatores básicos que limitam a taxa das comunicações [13].

A fim de obter uma maior confiabilidade na transmissão de informação através de canais ruidosos duas estratégias podem ser adotadas: 1) construir circuitos com componentes mais confiáveis e aumentar a potência dos sinais transmitidos; e 2) codificar a informação a ser transmitida, inserindo redundância de maneira controlada, reduzindo assim os efeitos danosos do ruído sobre a mensagem original. Na Figura 1.1 encontra-se ilustrado um modelo básico de sistema de comunicações codificado, ou seja, que utiliza a segunda abordagem. Nele, o codificador transforma a mensagem original **s** em uma mensagem codificada **t**, adicionando redundância à mensagem original, a qual é enviada através de um canal ruidoso para o decodificador. O canal possivelmente adiciona ruído à mensagem transmitida, gerando uma mensagem recebida **r**. O decodificador então usa a redundância introduzida pelo codificador para recuperar a mensagem original **s** a partir de **r**. As bases para o

desenvolvimento de bons códigos corretores de erros foram introduzidas por Shannon [14], o qual estabeleceu os limites teóricos da transmissão de informação codificada através de um canal ruidoso, embora não tenha mostrado como construir tais códigos, fato este que tem motivado muito estudo ao longo de todos os anos desde a publicação do referido artigo.

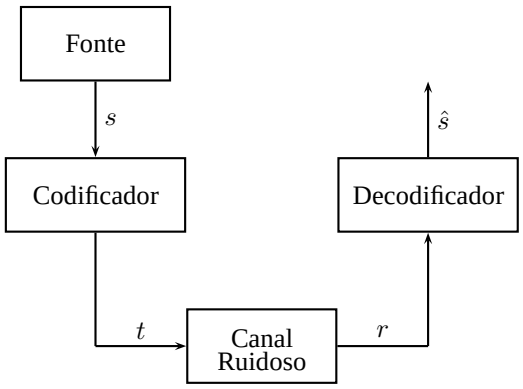


Figura 1.1: Sistema de comunicações codificado.

A introdução dos conceitos acima se faz importante no contexto da presente dissertação porque canais com apagamento são um tipo de canal ruidoso. Fazendo-se uso de bons códigos corretores de erros, a comunicação através de canais que corrompem a informação transmitida pode ser vista como uma transmissão em um canal com apagamentos, ou seja, caso um pacote de informação contenha erros detectáveis porém incorrigíveis pelo código corretor em uso, o mesmo é descartado e interpretado como um apagamento. Essa estratégia é essencial para a comunicação utilizando códigos LT, visto que os mesmos não são utilizados para detectar ou corrigir erros nas mensagens transmitidas.

1.2 Comunicação em canais com apagamento

Introduzidos por Elias [15], canais com apagamento são canais em que a mensagem é recebida sem erro ou é convertida em um apagamento. Conforme dito anteriormente, canais ruidosos nos quais bons códigos corretores de erros são aplicados podem ser vistos como canais com apagamento. Na transmissão de pacotes formados por um número fixo *m* de *bits*, o canal com apagamento formado por um alfabeto não-binário GF(q), em que $q = 2^m$, é um modelo adequado para canais nos quais tais pacotes podem ser descartados, sendo cada pacote representado por um dos $q = 2^m$ elementos de

GF(q). Um modelo de canal com apagamento é mostrado na Figura 1.2. Nela encontra-se ilustrado um canal quaternário ($q = 2^2$) com apagamento, o qual possui uma probabilidade $1 - p$ de transmitir o símbolo de entrada sem erro, e uma probabilidade p de entregar ao receptor o símbolo '$\triangle$', o qual significa um apagamento. Conforme dito anteriormente, transmissões realizadas através de redes de comunicação formam um importante exemplo de comunicação através de canais com apagamento.

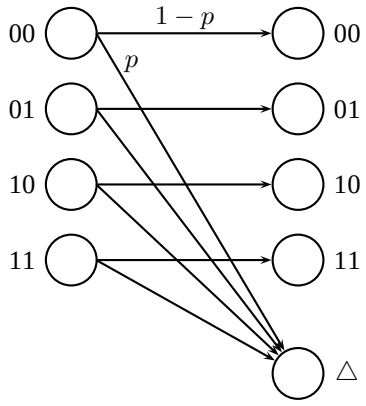


Figura 1.2: Canal quaternário com apagamento.

Os códigos de bloco [8] tradicionalmente utilizados para canais com apagamento são os códigos Reed-Solomon [16]. Um código Reed-Solomon (n, k) é capaz de recuperar a mensagem original a partir de qualquer conjunto de k símbolos codificados. No entanto, apesar do processo de codificação poder ser realizado de forma rápida, a complexidade do processo de decodificação torna estes códigos viáveis apenas para pequenos valores de k e n, códigos Reed-Solomon tipicamente utilizam $n = 255$ [2, 8, 17]. Além disso, ao contrário do que acontece com códigos LT, deve-se estimar a probabilidade de apagamento do canal antes da codificação de modo a determinar a taxa ($R = k/n$) do código. No capítulo 2 é feita uma breve comparação entre dois códigos para canais com apagamento e os códigos LT.

1.3 Fontes digitais

O conceito de fonte digital introduzido em [4] surgiu da intenção de se desenvolver um esquema de codificação no qual, a partir de um subconjunto qualquer de pacotes codificados cuja cardinalidade seja igual a do conjunto de pacotes que constituem os dados originais, o receptor seja capaz de recuperar a mensagem originalmente transmitida. A motivação para o nome fonte digital surgiu a partir da observação de que o codificador pode ser visto como uma fonte que provê um número

arbitrariamente grande de gotas de água (pacotes codificados[1]). Alguém que queira saciar sua sede com um copo d'água (recuperar o arquivo original) pode coletar um conjunto qualquer de gotas (símbolos codificados) que preencham o copo (qualquer conjunto de símbolos codificados com o mesmo número de símbolos do arquivo original). Por exemplo: suponha que o arquivo original é formado por $k.m$ *bits* e cada símbolo de entrada seja formado por m *bits* codificados. Um indivíduo que queira receber o arquivo original completo através de um canal de comunicação, deve receber k símbolos de saída até poder fazer a decodificação completa com sucesso [18].

Mais precisamente uma fonte digital ideal deve ter as seguintes propriedades:

- ▷ Possa gerar um número arbitrariamente grande de símbolos codificados a partir dos dados originais. É desejável ainda que, dada a mensagem original, cada símbolo codificado pode ser formado com um tempo constante.
- ▷ Um receptor seja capaz de reconstruir a mensagem original formada por k símbolos de entrada a partir de qualquer conjunto de k símbolos codificados. Essa reconstrução deve ser rápida, de preferência linear em k.

Foi citado anteriormente que códigos LT são capazes de recuperar, com probabilidade no mínimo $(1-\delta)$, um conjunto de k símbolos de entrada a partir de quaisquer $k + O(\sqrt{k}.\ln^2(k/\delta))$ símbolos codificados, necessitando para isso de uma média de $O(k.\ln(k/\delta))$ operações "ou-exclusivo". Conforme será detalhado posteriormente, diz-se que uma função $f(x)$ é $O(g(x))$ se, para algum $x > x_0$, ocorre que $f(x) \leq M.g(x)$ para uma constante M qualquer, ou seja, $g(x)$ representa uma cota superior para os valores de $f(x)$. Assim, desconsiderando constantes M, o número de operações e o número de símbolos necessários à decodificação de códigos LT são limitados, respectivamente, por: $k.\ln(k/\delta)$ e $k + O(\sqrt{k}.\ln^2(k/\delta))$. A Figura 1.3 ilustra tais limitantes, para $\delta = 0.5$, juntamente com o ideal linear.

Sendo assim, os códigos LT formam uma boa aproximação para o conceito de fontes digitais, em que o número de pacotes necessários à decodificação bem sucedida aproxima-se de k, com tempos de codificação e decodificação um pouco maiores do que o ideal linear. Ainda assim, são aproximações notáveis e eficientes para aplicações nas quais fontes digitais são desejadas.

[1] Ao longo do documento usaremos alternadamente os termos símbolos de saída, símbolos codificados, símbolos código e pacotes de saída para denominar um conjunto de *bits* que foram submetidos ao algoritmo de codificação dos códigos LT. De forma análoga, os termos símbolo de entrada ou pacote de entrada irão denominar um conjunto de *bits* do arquivo original a ser submetido ao referido algoritmo.

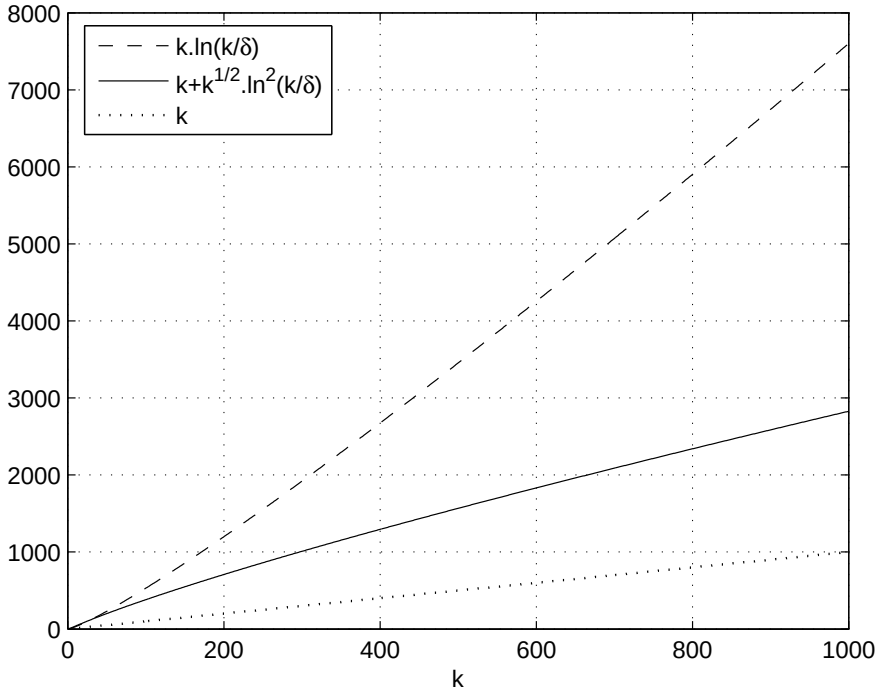


Figura 1.3: Limitantes de complexidade envolvidos na análise dos códigos LT.

1.3.1 Aplicações de fontes digitais

Existem inúmeras aplicações onde fontes digitais são desejáveis. A motivação desta subseção consiste em citar algumas delas para enfatizar a importância do estudo da classe de códigos de que trata essa dissertação.

Multicast

Por *multicast* subentende-se a transmissão em que uma fonte envia arquivos para um conjunto de determinados receptores em uma rede de comunicação. A principal vantagem da utilização de fontes digitais em transmissões do tipo *multicast* é a eliminação da necessidade de retransmissão. Se um usuário deixa de receber um determinado pacote, ele não precisa pedir retransmissão do mesmo visto que, qualquer conjunto de símbolos de saída formado pelo mesmo número de símbolos de entrada que formam o arquivo original pode ser utilizado para recuperar o arquivo enviado.

Uma outra vantagem das fontes digitais é que se torna trivial lidar com receptores operando a diferentes taxas de recepção. O transmissor pode enviar os dados na taxa mais alta possível, pois não haverá prejuízo algum para o receptor mais lento se este perder alguns pacotes. Fica claro também que iniciar *downloads* em insantes posteriores ao início da transmissão torna-se muito simples,

visto que a ordem na qual os pacotes são recebidos não importa (ao contrário de sistemas que usam protocolos TCP [1]).

Recepção paralela de arquivos

A recepção em paralelo de arquivos vindos de várias fontes distintas é outra aplicação em que o conceito de fontes digitais mostra-se extremamente útil. Usando fontes digitais, cada fonte pode produzir uma seqüência infinita de pacotes codificados; como a disponibilidade de pacotes codificados é essencialmente infinita, a possibilidade de colisão (quando o mesmo pacote é enviado por múltiplas fontes) é ínfima. Além disso, pode-se encerrar uma seção de recepção sem preocupação alguma com as fontes ou com a taxa de perdas do canal através do qual cada uma delas está enviando seus pacotes de saída.

Transmissão de vídeo em tempo real

Enquanto a transmissão de um vídeo inteiro é feita como a de qualquer arquivo, a transmissão em tempo real é algo que demanda uma análise mais profunda devido ao atraso envolvido na codificação e decodificação de arquivos usando fontes digitais. Usando uma abordagem comum, o usuário teria de esperar que todos os pacotes necessários à decodificação chegassem para começar a assistir ao vídeo.

Para transmissão em tempo real, a abordagem mais utilizada consiste em segmentar o arquivo original em várias partes. Enquanto a primeira parte é executada a segunda é baixada e assim por diante. Existem muitos aspectos a serem levados em consideração nessa abordagem como o tamanho relativo dos segmentos, a razão entre a taxa de *download* e a taxa de execução etc. Detalhes sobre esse assunto encontram-se minuciosamente descritos em [19].

1.4 Objetivos

Esta dissertação apresenta um estudo sobre códigos LT, suas ferramentas de análise assintótica, além de técnicas para obter boas distribuições de graus. Boas distribuições de graus são aquelas que geram códigos LT com baixa probabilidade de falha ao longo da decodificação. É proposto ainda um método de geração de cabeçalhos de pequeno tamanho para códigos LT que elimina a necessidade de sincronismo entre o codificador e o decodificador, simplificando assim, o processo de decodificação. Estes cabeçalhos informam ao decodificador quantos e quais são os vizinhos de cada

símbolo codificado recebido, visto que esta informação é essencial à implementação do processo de decodificação.

1.5 Organização da dissertação

Esta dissertação encontra-se organizada em cinco capítulos e dois apêndices. As referências bibliográficas localizam-se nas últimas páginas do texto e estão organizadas na ordem em que foram citadas. Segue abaixo um resumo de cada capítulo.

- **Capítulo 2.** Apresentam-se os códigos LT, seu processo de codificação e decodificação, análise de desempenho em conjunto com uma comparação a códigos tradicionais para canais com apagamento e um método para geração de cabeçalhos que possibilita a implementação de uma transmissão assíncrona.
- **Capitulo 3.** Descreve-se técnicas de análise assintótica de códigos LT juntamente com o cálculo da probabilidade de falha no processo de decodificação.
- **Capitulo 4.** São apresentadas duas distribuições de graus para códigos LT, além de uma técnica para construir boas distribuições de graus.
- **Capítulo 5.** Conclusões e sugestões de trabalhos futuros são discutidas.
- **Apêndice A.** Descreve a teoria básica de cotas de Chernoff.
- **Apêndice B.** Neste apêndice são revistos conceitos importantes de teoria de grafos utilizados ao longo da dissertação.

CAPÍTULO 2

CÓDIGOS LT

A descoberta consiste em ver o que todos viram e pensar no que ninguém pensou.

— Szent-Gyorgi

Os códigos Luby Transform (LT) formam uma classe de códigos criada por Michael Luby em 2002 [6], sendo estes a primeira implementação do conceito de fonte digital (*digital fountain*) com tempo de codificação e decodificação viável para sistemas que requeiram a formação de um grande número de símbolos codificados n a partir de um arquivo com k símbolos de entrada. O objetivo deste capítulo consiste no estudo dessa recém-criada classe de códigos, suas propriedades e possíveis aplicações.

Como primeira implementação do conceito de fonte digital para arquivos de tamanho prático, os códigos LT são capazes de recuperar, com probabilidade no mínimo $(1 - \delta)$, um conjunto de k símbolos de entrada a partir de quaisquer $k+O(\sqrt{k}.\ln^2(k/\delta))$ símbolos codificados, necessitando de uma média de $O(k.\ln(k/\delta))$ operações [6]. Os códigos LT são classificados como sem-taxa, sendo o número de símbolos codificados que podem ser gerados a partir dos dados originais potencialmente ilimitado[2]. Assim, não importa qual o modelo de perdas do canal em uso, o codificador simplesmente envia dados até um número suficiente de pacotes chegarem ao receptor. A estatística do canal apenas influencia no tempo necessário para que símbolos de saída suficientes para a decodificação bem sucedida sejam recebidos pelo decodificador. Como o decodificador pode recuperar uma mensagem formada por k símbolos de entrada a partir de quaisquer $(1+\epsilon)k$ símbolos codificados onde $0 < \epsilon <$

[2]Na verdade, apesar do termo códigos sem-taxa ter sido largamente adotado na literatura, os códigos LT são códigos de taxa variável, sendo o número de símbolos codificados (n) variável de acordo com o número de pacotes necessários à decodificação bem sucedida.

1, os códigos LT são quase ótimos com respeito a qualquer canal com apagamentos, considerando que um código ótimo é aquele que consegue recuperar a mensagem original formada por k símbolos de entrada, a partir de qualquer subconjunto de k símbolos de saída dentre os n gerados [9].

2.1 Codificação

Nesta seção será abordado o processo de codificação de arquivos utilizando códigos LT. Ao longo da codificação cada símbolo codificado t_n é obtido a partir de um arquivo original formado pelos símbolos $s_1, s_2, s_3, ..., s_k$ como segue:

1. Aleatoriamente escolher o número de vizinhos (grau) d_n do símbolo codificado a partir de uma distribuição de probabilidades $\Omega(x)$.
2. Escolher, uniforme e aleatoriamente, d_n símbolos de entrada distintos e atribuir a t_n o valor da adição *bit* a *bit*, módulo 2 desses d_n símbolos.

Fica claro a partir da descrição do processo que existem duas distribuições de probabilidade envolvidas. Uma para determinar o número de vizinhos (grau) de cada símbolo codificado, e outra, uniforme, para reger a escolha de quais símbolos de entrada irão compor o conjunto de vizinhos do símbolo codificado. A distribuição de probabilidade que rege o número de vizinhos de um símbolo codificado é definida como se segue:

Definição 2.1 *Para todo d, Ω_d é a probabilidade que um símbolo codificado tenha grau d.*

Ao longo dessa dissertação, denotar-se-á uma distribuição de probabilidades pelo seu polinômio gerador. Assim, a representação para a distribuição de graus é feita pelo polinômio $\Omega(x) = \sum_{i=1}^{k} \Omega_i x^i$, onde Ω_i denota a probabilidade de que o grau i seja escolhido. Usando essa notação, por exemplo, o valor esperado da distribuição de graus é dado simplesmente por $\Omega'(1)$, onde $\Omega'(x)$ denota a derivada de $\Omega(x)$ com respeito a x.

Faz-se mister lembrar que para realizar a decodificação, o receptor precisa conhecer o grau do símbolo codificado (*d*) e quais são os vizinhos do mesmo [3]. Existem diversas maneiras de comunicar ao receptor tais dados conforme descrito na Seção 2.3 e em [6, 20]. Na implementação utilizada no desenvolvimento desta dissertação, optou-se por um esquema assíncrono para comunicar tais dados,

[3] Pode-se argumentar que conhecendo os vizinhos não há necessidade de se enviar também o grau do símbolo codificado. Isso depende da implementação computacional dos algoritmos de codificação e decodificação. Nas implementações realizadas no desenvolvimento dessa dissertação, conhecer o grau de cada símbolo faz parte do algoritmo de determinação dos vizinhos, conforme exposto na Seção 2.3.2. Logo, o grau de cada símbolo de saída precisa ser enviado.

conforme exposto na Seção 2.3. A operação de codificação define um grafo ligando os pacotes de entrada e os pacotes de saída, e é baseado nessa descrição que será explanado o processo de decodificação. Descrito o processo de codificação, ficará clara a representação de códigos LT através de grafos, a partir do seguinte exemplo:

Exemplo 1: *Considere um arquivo formado por três "bits"* $s_1 s_2 s_3 = 101$*, perceba que por simplicidade os símbolos de entrada neste exemplo são formados por apenas 1 "bit". O número de "bits" de um símbolo de entrada é arbitrário, podendo ser definido de acordo com a aplicação em vista. Quer-se realizar a codificação desse arquivo gerando quatro símbolos de saída* $t_1 t_2 t_3 t_4$*. Assim, o processo de codificação será composto de quatro etapas, uma para cada símbolo de saída gerado. A Tabela 2.1 ilustra o resultado da codificação para cada símbolo de saída. Note que o símbolo* $\oplus$ *denota a operação "ou-exclusivo" componente a componente.*

Tabela 2.1: Processo de codificação do Exemplo 1.

símbolo	grau $(d(t_i))$	vizinhos	valor
t_1	1	s_1	$s_1 = 1$
t_2	3	$s_1 s_2 s_3$	$s_1 \oplus s_2 \oplus s_3 = 0$
t_3	2	$s_2 s_3$	$s_2 \oplus s_3 = 1$
t_4	2	$s_1 s_2$	$s_1 \oplus s_2 = 1$

Para construir um grafo que retrate tal codificação vamos seguir a seguinte notação: os símbolos codificados (t_n*) formarão os nós de verificação e os símbolos de entrada (*s_k*) formarão os nós de mensagem de um grafo bipartite (Apêndice B). Cada símbolo de saída será conectado através de um ramo ao símbolo de entrada que for seu vizinho. Desta forma, apresentamos na Figura 2.1 o seguinte grafo que ilustra a codificação.*

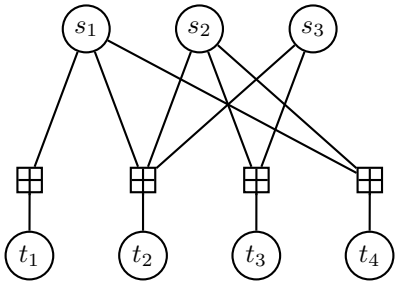


Figura 2.1: Grafo resultante da codificação realizada no Exemplo 1.

2.2 Decodificação

Relembrando o acima exposto, vamos adotar a seguinte notação: os símbolos codificados (t_n) formarão os nós de verificação e os símbolos de entrada (s_k) formarão os nós de mensagem. O algoritmo de decodificação de códigos LT resume-se ao seguinte:

1. Encontrar um nó de verificação t_n que esteja conectado a apenas um símbolo de entrada s_k, caso não haja tal nó de verificação, a decodificação é abortada, sendo necessário receber mais símbolos codificados antes de se fazer nova tentativa de decodificação.

   a) Fazer $s_k = t_n$,

   b) Somar s_k a todos os nós t'_n que estejam conectados a s_k,

   c) Remover todas as conexões que chegam ao símbolo de entrada s_k.

2. Repetir (1) até que todos os s_k estejam determinados.

O processo de decodificação descrito está ilustrado na Figura 2.2 na qual cada símbolo de entrada é composto de apenas um *bit*. Existem três símbolos de entrada (indicados pelos nós de símbolo superiores) e quatro símbolos de saída (indicados pelos nós de verificação inferiores), os quais valem $t_1t_2t_3t_4 = 1011$ no início do algoritmo. Na primeira iteração, o único nó de verificação conectado a apenas um símbolo de entrada é t_1 (a). Faz-se então $s_1 = t_1$ (b), elimina-se o nó de verificação, e adiciona-se o valor de s_1 aos nós de verificação que estão ligados ao mesmo (c) desconectando s_1 do grafo. No início da segunda iteração (c), o quarto nó de verificação está ligado apenas a s_2. Faz-se então $s_2 = t_4$ (d), e se adiciona s_2 aos dois nós de verificação ligados a ele (e). Por fim, percebe-se que dois nós de verificação estão ligado a s_3, e ambos atribuem o mesmo valor a tal símbolo de entrada, o qual é restaurado em (f).

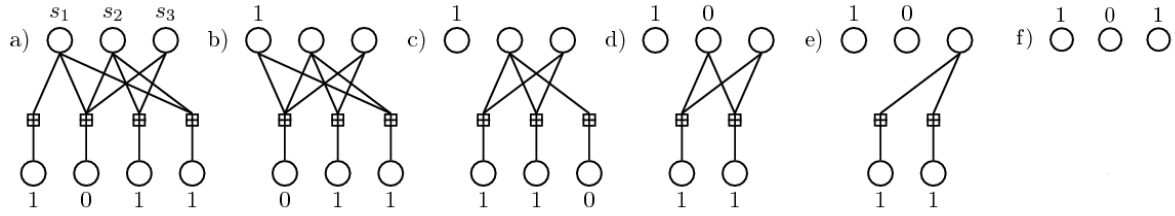


Figura 2.2: Decodificação dos símbolos gerados no Exemplo 1.

2.2.1 O processo LT

A descrição do que se define por processo LT é apenas uma ferramenta para descrever o projeto e análise de uma boa distribuição de graus. O processo LT é uma generalização do clássico problema de lançar bolas em um conjunto de urnas e muitos dos resultados desse problema em teoria da probabilidade serão aplicados na análise do código em questão. Na análise subjacente, símbolos codificados são análogos a bolas, e símbolos de entrada são análogos a urnas.

Processo LT: *Todos os símbolos de entrada estão inicialmente descobertos. Um símbolo de entrada é dito coberto, se ao longo da decodificação recebe-se um símbolo de grau unitário que o tem como vizinho. No primeiro passo do processo LT, todos os símbolos codificados com apenas um vizinho são liberados para cobrir o mesmo. O conjunto de símbolos de entrada cobertos os quais não foram ainda processados é chamado de "ripple". Em cada passo subseqüente do processo LT, um símbolo de entrada no "ripple" é processado, ou seja, ele é removido como vizinho de todos os símbolos codificados que o tenham como tal; após isso, todos os símbolos codificados que possuírem apenas um vizinho são liberados para cobrir tal vizinho restante. Alguns destes podem ser símbolos anteriormente descobertos fazendo com que o "ripple" cresça, enquanto outros que já foram cobertos não causam qualquer alteração no referido "ripple". O processo termina quando o "ripple" encontrar-se vazio em algum passo. O processo falha quando existir ao menos um símbolo de entrada não-coberto ao fim do processo, e é bem sucedido se ao fim do mesmo todos os símbolos de entrada são cobertos.*

A Figura 2.3 ilustra o processo LT para a codificação realizada no Exemplo 1. Inicialmente (a) todos os símbolos de entrada (urnas) estão descobertos (urnas sem bolas). No primeiro passo todos os símbolos codificados (representados por retângulos, as bolas dentro dos retângulos representam os vizinhos do símbolo codificado) com apenas um vizinho são liberados para cobrir o mesmo (b). Em cada passo subseqüente, um símbolo de entrada do *ripple* é processado, ou seja, ele é removido como vizinho de todos os símbolos codificados que o tenham como tal, os símbolos sombreados em (c), (e) e (g) indicam tal processamento. Em (c) o *ripple* é formado por s_1, em (e) por s_2 e em (g) por s_3, ou seja, o conjunto de símbolos de entrada cobertos mas ainda não processados. O processo termina quando o *ripple* encontrar-se vazio em algum passo (h). O processo falha quando existir ao menos um símbolo de entrada não-coberto ao fim do processo, e é bem sucedido se ao fim do mesmo todos os símbolos de entrada são cobertos. No nosso exemplo o processo foi bem sucedido.

No processo clássico de lançamento de n bolas em um conjunto de de k urnas, todas as bolas são lançadas de uma única vez, tornando necessário um grande número de bolas para cobrir todas as

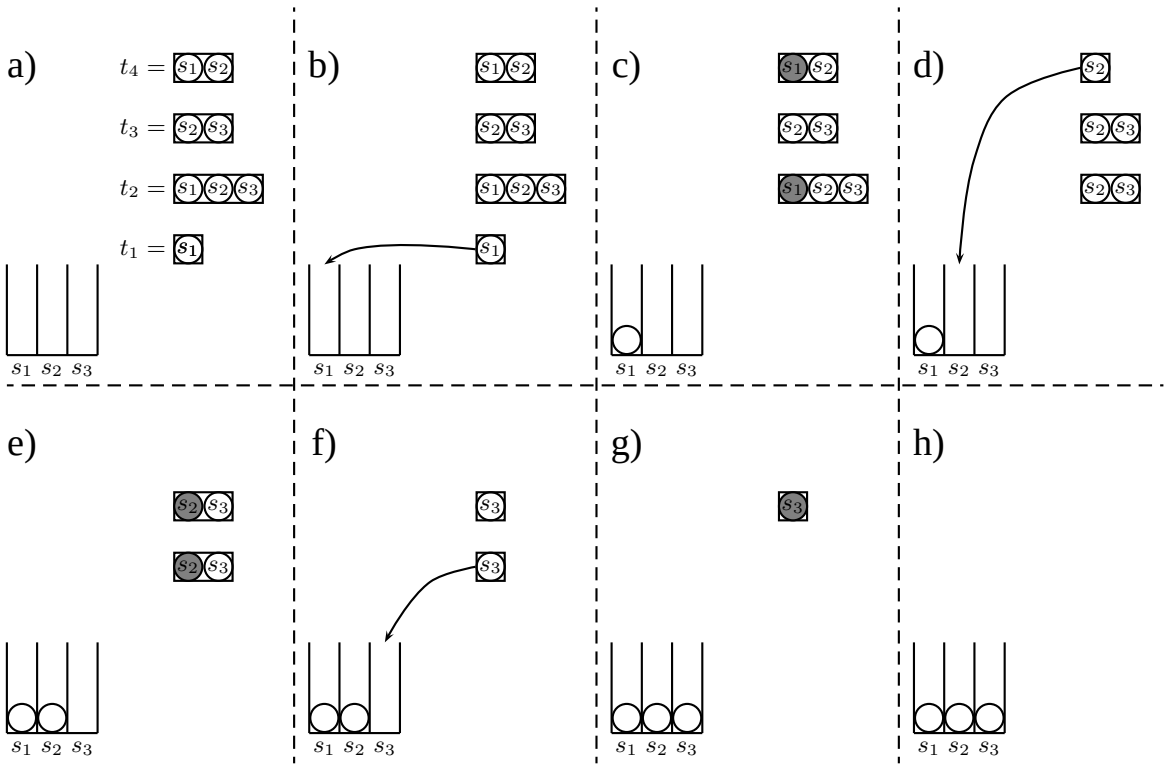


Figura 2.3: Processo LT.

urnas, mais precisamente, se n bolas são lançadas em k urnas, o valor esperado do número de urnas vazias é dado por $k.e^{n/k}$ [7]. Um bom projeto de uma distribuição de graus garante que o processo LT libera símbolos codificados de forma incremental para cobrir todos os símbolos de entrada. Os objetivos de uma boa distribuição de graus são:

1. Liberar símbolos a uma taxa tal que o *ripple* não cresça exageradamente ao longo do processo (o que geraria um desperdício de recursos computacionais visto que ao serem liberados, os símbolos codificados não irão cobrir nenhum símbolo de entrada ainda não-coberto).
2. Não permitir que o *ripple* desapareça antes de todos os símbolos de entrada estarem devidamente cobertos.

Não é muito difícil verificar uma estreita correspondência entre o processo LT e o processo de decodificação de um código LT. O número total de símbolos codificados necessários para cobrir todos os símbolos de entrada no processo LT corresponde ao número total de símbolos codificados necessários para recuperar os dados de entrada no processo de decodificação. A soma dos graus dos símbolos codificados corresponde exatamente ao número total de operações necessárias para recuperar a mensagem original.

Na seção seguinte, iremos fazer uma análise do desempenho dos códigos LT em canais livres de ruído, canais com apagamento e canais binários simétricos (BSC). Para tanto, vamos introduzir agora a definição de duas distribuições de graus utilizadas no processo de codificação de códigos LT clássicos. Dentre elas, a distribuição Sóliton Robusta é a utilizada na aplicações práticas dos códigos LT [6, 20].

Definição 2.2 *A distribuição Sóliton Ideal* $\rho(x) = \sum_{d=1}^{k} \rho_d x^d$ *é definida por* $\rho_1, ..., \rho_k$*, em que:*

▷ $\rho_1 = 1/k$.

▷ *Para todo* $d = 2, ..., k, \rho_d = 1/d(d-1)$.

Definição 2.3 *A distribuição Sóliton Robusta* $\mu(x) = \sum_{d=1}^{k} \mu_d x^d$ *é definida como se segue. Seja* $R = c.\ln(k/\delta)\sqrt{k}$ *para alguma constante* $c > 0$ *e seja* τ_d *tal que:*

$$\tau_d = \begin{cases} R/d.k & para\ d = 1, ..., k/R - 1, \\ R.\ln(R/\delta)/k & para\ d = k/R, \\ 0 & para\ d = k/R + 1, ..., k. \end{cases}$$

Adicionando ρ_d *a* τ_d *e normalizando esta soma dividindo-a por* $\beta = \sum_{d=1}^{k}[\rho_d + \tau_d]$ *tem-se* μ_d*, ou seja:*

$$\mu_d = (\rho_d + \tau_d)/\beta \text{ para } d = 1, ..., k, .$$

O parâmetro c é uma constante de ordem 1. No entanto, valores menores que 1 geram bons resultados. Nesta dissertação, o número de símbolos codificados é definido como $n = \beta k = (1+\epsilon)k$. O número de símbolos codificados de grau d é uma variável aleatória discreta de distribuição de probabilidade binomial com parâmetros n e μ_d. O valor esperado de uma variável binomialmente distribuída X com parâmetros n e p é igual a $n.p$ [21]. Logo, o valor esperado do número de símbolos codificados de grau d é dado por:

$$n.\mu_d = k\beta.\frac{(\rho_d + \tau_d)}{\beta} = k.(\rho_d + \tau_d).$$

No próximo capítulo serão analisados os requisitos que uma boa distribuição de graus deve satisfazer para que a probabilidade de sucesso na decodificação seja maximizada, justificando assim o porquê da escolha da distribuição Sóliton Robusta.

2.3 Formação de cabeçalhos para códigos LT

Esta seção apresenta as idéias envolvidas na geração do cabeçalho inerente à utilização dos códigos LT. É fato conhecido que tais códigos necessitam que sejam enviados, juntamente com cada um dos símbolos codificados, informações referentes a como os mesmos foram gerados (quantos e quais são os seus vizinhos). A proposta a seguir visa minimizar o número de *bits* necessários ao envio de tais informações, bem como eliminar a por vezes complicada necessidade de sincronismo entre transmissor e receptor.

2.3.1 Método das transformações inversas para variáveis aleatórias discretas

Caso se queira simular uma variável aleatória discreta X a qual pode assumir k valores distintos, com distribuição de probabilidade

$$P\{X = x_j\} = P_j \text{ para } j = 1, ..., k \text{ e } \sum_j P_j = 1,$$

pode-se usar a versão de tempo discreto da técnica da transformação inversa exposta a seguir, conforme em [21].

Para simular a variável aleatória discreta X para a qual $P\{X = x_j\} = P_j$, considere U uma variável aleatória uniformemente distribuída entre $(0, 1)$, e faça

$$X = \begin{cases} x_1, & se\, U < P_1, \\ x_2, & se\, P_1 \leq U < P_1 + P_2, \\ \vdots & \\ x_j, & se\ \sum_{i=1}^{j-1} P_i \leq U < \sum_{i=1}^{j} P_i, \\ \vdots & \end{cases} \tag{2.1}$$

Como

$$P\{X = x_j\} = P\left\{\sum_{i=1}^{j-1} P_i \leq U < \sum_{i=1}^{j} P_i\right\} = P_j,$$

vemos que X tem a distribuição de probabilidade desejada. Uma abordagem mais detalhada do método em questão pode ser encontrada em [22].

A utilidade do método acima exposto na implementação do codificador e do decodificador LT vem da necessidade de se implementar, na geração de cada símbolo, experimentos os quais utilizam duas distribuições de probabilidade distintas: a distribuição Sóliton Robusta e a distribuição uniforme.

2.3.2 Implementação computacional da codificação

O problema computacional da codificação consiste em como implementar a determinação do grau dos símbolos codificados e seus respectivos vizinhos. Um aspecto importante diz respeito ao número de *bits* dos símbolos de entrada. Esse valor pode ser ajustado conforme a aplicação, variando desde um a milhares de *bits*. Por exemplo, em um canal da Internet baseado em pacotes, a carga útil de um pacote IP (*Internet Protocol*) é de 1024 *bytes* [1]. Logo, assumindo que cada pacote IP contenha em sua carga útil um símbolo de saída e 9 *bytes* de informação auxiliar (cabeçalho), um símbolo de entrada contendo $M = (1024 - 9)$ *bytes*, ou seja, 8120 *bits* seria apropriado.

Um outro exemplo é um sistema fazendo uso do padrão de pacotes MPEG (*Moving Picture Expert Group*) [1] o qual possui uma carga útil de 188 *bytes* em cada pacote. Considerando que cada pacote contenha um símbolo de saída e 8 *bytes* de cabeçalho, símbolos de entrada contendo $M = (188 - 8)$ *bytes*, ou seja, 1440 *bits*, seriam adequados.

Determinação do grau do símbolo codificado

Usando o método das transformações inversas, a determinação do grau dos vizinhos a partir de um número uniformemente distribuído entre 0 e 1 é trivial. Basta adotar $x_j = j$ onde j determina o grau do símbolo codificado e $P_j = \Omega_j$.

Conforme exposto anteriormente, o primeiro passo da codificação consiste na determinação do grau do símbolo a ser codificado. Assim sendo, utiliza-se o método das transformações inversas para simular a distribuição Sóliton Robusta $\mu(x)$, armazenando para cada símbolo codificado um número U_1 uniformemente distribuído entre $(0, 1)$. Perceba que de posse de U_1 e dos possíveis valores para o grau de um símbolo codificado, o decodificador pode aplicar o mesmo método das transformações inversas e obter univocamente o grau do símbolo codificado ao qual U_1 está associado, como mostra o exemplo a seguir.

Exemplo 2: *Considere uma variável aleatória discreta X para a qual* $P\{X = x_j\} = P_j$ *em que* $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, $x_4 = 4$ *e* $P_1 = 0.1$, $P_2 = 0.5$, $P_3 = 0.2$, $P_4 = 0.2$. *Para simular a distribuição de probabilidades acima usando o método das transformações inversas, amostra-se uma variável uniformemente distribuída entre 0 e 1. Seja* U_1 *o resultado desse experimento, na Tabela 2.2 encontra-se o valor atribuído à variável aleatória* X *para os possíveis valores de* U_1.

Tabela 2.2: Método das transformações inversas para simular a variável aleatória X do Exemplo 2.

U_1	X
$U_1 < 0.1$	1
$0.1 \leq U_1 < 0.1 + 0.5$	2
$0.1 + 0.5 \leq U_1 < 0.1 + 0.5 + 0.2$	3
$0.1 + 0.5 + 0.2 \leq U_1 < 0.1 + 0.5 + 0.2 + 0.2$	4

Determinação dos vizinhos

Para determinar os vizinhos de cada símbolo procede-se da seguinte maneira descrita em [20, 23]. De posse do grau (d) do símbolo de saída em questão e do número de símbolos de entrada, gera-se de forma uniforme dois números X e Y. É importante ressaltar que $1 \leq X \leq k-1$ e $0 \leq Y \leq k-1$. Tais números podem ser gerados através do método das transformações inversas, simulando uma distribuição uniforme entre $(1, k-1)$ para determinar X a partir de um número U_2 uniformemente distribuído entre $(0, 1)$; e simulando uma distribuição uniforme entre $(0, k-1)$ a partir de outro número U_3 uniformemente distribuído entre $(0, 1)$ para determinar Y. Feito isto, é criado um vetor AL$[\cdot]$ o qual irá conter todos os vizinhos do símbolo código em questão. Assim, o seguinte algoritmo é especificado:

Para $j = 1, ..., d-1$:

1. $AL(0) = Y$,
2. $AL(j) = (AL(j-1) + X) mod(k)$.

É importante ressaltar que, de modo a fazer que todos os vizinhos sejam distintos, k deve ser primo. Assim, se o número de símbolos de entrada não for primo, devem ser inseridos tantos símbolos de entrada quanto necessários para que k se torne primo, sendo que estes símbolos adicionais devem ser formados apenas por *bits* de valor 0. Satisfeita esta condição, e como o grau $d \leq k$ todos os vizinhos do símbolo de saída serão distintos. Em [24], pode ser encontrada a descrição de um método o qual dispensa a condição de primalidade de k, no entanto, o esquema de formação de cabeçalhos descrito aqui não pode ser utilizado caso se opte pela utilização de tal método.

Por fim, tudo o que o decodificador necessita saber de modo a recuperar X e Y, e por conseguinte os vizinhos do símbolo codificado em questão, são os números U_2 e U_3, juntamente com o algoritmo de formação dos vizinhos exposto acima.

2.3.3 Formação do cabeçalho

Para que se possa realizar a decodificação de um símbolo de saída, o decodificador deve conhecer o grau e os vizinhos do símbolo codificado em questão. Os modos de se comunicar essa informação ao decodificador incluem:

1. Enviar de forma explícita ao decodificador, o grau e uma lista dos vizinhos para cada símbolo codificado enviado.
2. Calcular implicitamente, baseado em um sincronismo do sistema, o grau e os índices dos vizinhos de cada símbolo de saída.
3. Associar um cabeçalho a cada símbolo codificado de forma que ambos, codificador e decodificador, apliquem uma mesma função ao cabeçalho para produzir o grau e o conjunto de vizinhos do símbolo codificado.

A primeira alternativa implica em um grande número de *bits* para a sua representação. Já a segunda opção necessita de um sincronismo entre o codificador e o decodificador, um requisito por vezes difícil de ser implementado. Assim, optou-se pela terceira opção, a qual possibilita a formação de um cabeçalho com pequeno número de *bits* e ao mesmo tempo elimina a necessidade de sincronismo no sistema. Seguindo a idéia da terceira opção, foi desenvolvido e apresentado como contribuição desta dissertação, o seguinte esquema de geração de cabeçalhos.

A cada símbolo codificado será associado um cabeçalho formado pela concatenação da parte fracionária dos números U_1, U_2 e U_3 convertida para a representação binária, conforme mostra o exemplo a seguir.

Exemplo 3: *Considere que durante o processo de geração de um símbolo codificado, os seguintes valores foram determinados:* $U_1 = 0.4$, $U_2 = 0.3$ *e* $U_3 = 0.1$. *Perceba que nesse caso a precisão utilizada é de apenas uma casa decimal. Desta forma, seguindo o esquema de geração proposto, o símbolo de saída em questão teria o cabeçalho* $[100\ 011\ 001]$ *formado pela representação binária da parte fracionária de* U_1, U_2 *e* U_3 *(a separação entre os números binários é somente ilustrativa, em aplicações práticas tais números podem ser concatenados).*

O número de casas decimais necessárias para representar a parte fracionária dos números U_1, U_2 e U_3, de forma a garantir uma decodificação correta, varia com o número de símbolos de entrada do arquivo submetido à codificação conforme pode ser visto na Tabela 2.3. Por exemplo, para um arquivo de entrada subdividido em 10000 pacotes, 9 casas decimais são suficientes para uma decodificação

bem sucedida. Fazendo a conversão para a base binária teríamos a necessidade de aproximadamente 30 *bits* para cada um dos U´s, o que geraria um cabeçalho de $30 \cdot 3 = 90$ *bits*.

Pode-se perceber que essa precisão (número de casas decimais consideradas na representação de U_1, U_2 e U_3) depende do comprimento dos intervalos de decisão do método das transformações inversas. Visto que, as regiões de decisão (intervalos do método das transformações inversas) se tornam mais estreitas à medida que o número de símbolos de entrada aumenta, deve-se aumentar a precisão da chave quando se aumenta k, a fim de evitar erros de decodificação devidos a aproximações nos valores dos U's. Assim sendo, há uma estreita correspondência entre o número de símbolos de entrada e a precisão da chave, devendo esta ser determinada antes da codificação para se evitar erros oriundos de aproximações no valor dos U´s.

Um detalhe importante a ser apontado é que o tamanho dos pacotes não influencia na precisão do cabeçalho, ou seja, pacotes de 10 ou 10000 *bits* necessitam da mesma precisão se o número de símbolos de entrada for o mesmo. A Tabela 2.3 ilustra o número de casas decimais necessárias à representação de U_1, U_2 e U_3 em função do número de símbolos de entrada. Todas as simulações foram realizadas considerando símbolos com 5 *bits* de comprimento, tendo em vista que o número de *bits* dos símbolos de entrada não afeta a precisão dos números presentes no cabeçalho.

Considerando símbolos com mais de 1000 *bits*, a adição da chave proposta gera um acréscimo muito pequeno de informação auxiliar (em torno de 1%) em relação ao número de *bits* do símbolo. Pode-se inferir então que o esquema proposto é bem eficiente no que diz respeito à inserção de redundância aos símbolos codificados (símbolos de saída). Simulações em um canal livre de ruído (Tabela 2.3) mostraram que o esquema proposto é de fato eficiente, podendo ser implementado de forma rápida e simples, sem necessidade de sincronismo entre o codificador e o decodificador. Percebe-se o quão pequeno é esse cabeçalho quando se examina por exemplo um canal da Internet baseado em pacotes, nessa aplicação, um pacote com carga útil de 1024 *bytes* possui um número de *bits* apropriado. Assumindo que cada pacote contenha 8 *bytes* de informação auxiliar, um símbolo de entrada com $M = (1024 - 8) \cdot 8 = 8128$ *bits* seria apropriado. Considerando que o número de *bits* do símbolo de entrada seja igual ao número de *bits* do símbolo de saída, um cabeçalho de 90 *bits* representa apenas 1.1% do total de *bits* que formam um pacote de saída.

2.4 Análise de desempenho

É importante lembrar que para códigos LT, a probabilidade de sucesso na decodificação só é limitada se o número de símbolos de saída disponíveis ao receptor também for limitado. O que pode

Tabela 2.3: Precisão dos números do cabeçalho em função do número de símbolos de entrada.

número de símbolos de entrada	número de dígitos de precisão	número de *bits* do cabeçalho
100	6	60
1000	7	72
5000	8	81
10000	9	90
50000	10	102

ocorrer é um maior tempo de espera para a decodificação completa do arquivo, tendo em vista que para garantir o sucesso na decodificação pelo menos $k\beta$ símbolos codificados devem ser recebidos.

Nesta seção será analisado o desempenho de códigos LT construídos com a distribuição Sóliton Robusta. Pretende-se com isso avaliar a capacidade da distribuição de graus escolhida aproximar o código do conceito de fontes digitais; quão melhor a distribuição, melhor a aproximação. O estudo de códigos LT em canais com apagamento avalia o desempenho destes levando-se em consideração as características do canal, visto que símbolos de saída podem não permitir sua decodificação devido a apagamentos.

2.4.1 Desempenho em canais livres de ruído

É discutido aqui o número de pacotes necessários à decodificação de códigos LT em canais livres de ruído. As simulações foram implementadas considerando $k = 10000$ (número de símbolos de entrada). Ao longo do processo de decodificação, caso o receptor não obtivesse sucesso, o mesmo requisitava mais símbolos de saída ao transmissor. Tal procedimento foi realizado de forma a obter uma estimativa mais próxima do real número de símbolos necessários à decodificação bem sucedida. No caso de uma implementação prática, não existe necessidade de requisitar ao transmissor mais símbolos de saída, o decodificador começa a decodificação assim que estimar possuir pacotes suficientes para uma decodificação bem sucedida. No entanto, só deixa de aceitar símbolos de saída vindos do transmissor quando a informação original for recuperada, eliminando assim a necessidade de um canal reverso para requisitar mais símbolos de saída.

O número de símbolos extra requisitados a cada pedido (G) foi ajustado em 50, ou seja, a cada falha na decodificação, mais 50 símbolos de saída eram requisitados ao transmissor. Em [20], aconselha-se que $G = \sqrt{k}$. Nas simulações aqui descritas foi adotado o valor $G = \sqrt{k}/2$ a fim de obter uma maior precisão acerca do real número de pacotes necessários à decodificação bem sucedida. Na Tabela 2.4 encontram-se indicados o número médio de símbolos necessários à decodificação bem

sucedida, em função dos parâmetros c e δ.

Tabela 2.4: Número médio de símbolos necessários à decodificação em função de c e δ.

δ \ c	0.01	0.1	0.5	1.0
0.001	10538	12360	19271	22812
0.01	10495	11729	16850	20296
0.1	10456	11202	14740	17579
1	10423	10809	12991	15004

Cada uma das médias obtidas foi calculada após 30 simulações para cada par c e δ. O número de simulações foi escolhido de maneira arbitrária, tendo em vista que, o número símbolos de saída necessários para a recuperação da mensagem original, mostrou-se altamente concentrado em torno do valor médio exposto na Tabela 2.4. Uma questão pertinente é: qual o sentido de se operar com um $\delta = 1$, tendo em vista que este é uma cota da probabilidade de falha do decodificador. A utilização desse valor é justificada devido à análise probabilística utilizada para estabelecer δ como um limitante superior da probabilidade de falha na decodificação, ser muito pessimista [6, 18]. Dessa forma, quando se utiliza a distribuição Sóliton Robusta, δ é uma cota "frouxa" da probabilidade de falha, de forma que até valores como $\delta = 1$ são capazes de gerar bons códigos LT.

Ao longo da dissertação, os códigos LT utilizados farão uso da distribuição Sóliton Robusta com parâmetros c e δ iguais a 0.01 e 0.1 respectivamente, tendo em vista que se verificou, ao longo das simulações realizadas nessa dissertação, que estes valores geram bons códigos LT. Considera-se um bom código LT, aquele que é capaz de recuperar a mensagem original a partir de um número de símbolos de saída n que é um pouco ($\approx 5\%$) maior do que o número de símbolos de entrada que formam a mensagem original k.

2.4.2 Desempenho em canais com apagamento

O estudo do desempenho em canais com apagamento é de grande importância visto que, conforme dito anteriormente, um canal que pode corromper ou perder pacotes pode ser modelado como um canal com apagamento quando se trata da transmissão de pacotes em uma rede de comunicação. Para isso, assume-se que códigos corretores de erros (usados independentemente dos códigos em estudo) são usados para detectar e corrigir erros em um pacote, conforme ilustrado na Figura 2.4. Caso um dado pacote contenha mais erros do que a capacidade de correção do código corretor, a detecção de erros é usada para identificar o pacote que contém erros incorrigíveis, tratando os mesmos como

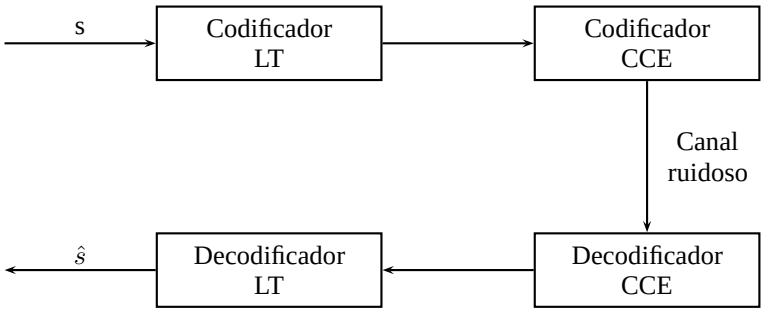


Figura 2.4: Sistema com códigos corretores de erros (CCE) usados em conjunto com códigos LT.

apagamentos. Apagamentos são descartados em seguida pelo decodificador LT. Assim, ou o pacote é recebido completamente sem erros ou é descartado.

A Figura 2.5 mostra a quantidade de símbolos necessários para uma decodificação bem sucedida em função da probabilidade de apagamento do canal. Percebe-se que, à medida que a probabilidade de apagamento cresce, maior se torna a quantidade de símbolos codificados necessários para que haja uma decodificação bem sucedida. Esse comportamento explica-se pelo fato de quão maior a probabilidade de apagamento, menos símbolos de saída chegam ao decodificador. É possível notar ainda que, com probabilidades de apagamento muito baixas, o número de pacotes necessários para a decodificação bem sucedida tende a uma constante igual a taxa mínima do código LT em uso, ou seja, o número mínimo de pacotes de saída necessários à decodificação. Isso acontece por que praticamente todos os pacotes enviados são recebidos pelo decodificador. É importante lembrar que ajustando os parâmetros c e δ pode-se obter códigos LT com taxas mínimas melhores do que a exibida pelo código utilizado nesta simulação, a qual vale aproximadamente 90%.

No caso de símbolos com apenas um *bit*, o canal reduz-se ao canal binário com apagamento (BEC) [15], no entanto, estamos considerando aqui símbolos de comprimento $l \geq 1$ *bits*, desta forma está-se considerando canais com apagamento q-ários, onde $q = 2^l$. As simulações foram feitas considerando $k = 1000$. O valor $(1 + \epsilon)k = \beta k$ representa o número de símbolos de saída necessários a uma decodificação bem sucedida.

2.4.3 Desempenho em canais BSC

A análise da probabilidade de falha do decodificador em canais BSC (*Binary Symmetric Channel*), pode ser reduzida ao caso do canal com apagamentos. Considere um canal BSC com uma probabilidade de erro p. Seja m o tamanho em *bits* do pacote que contém o símbolo codificado jun-

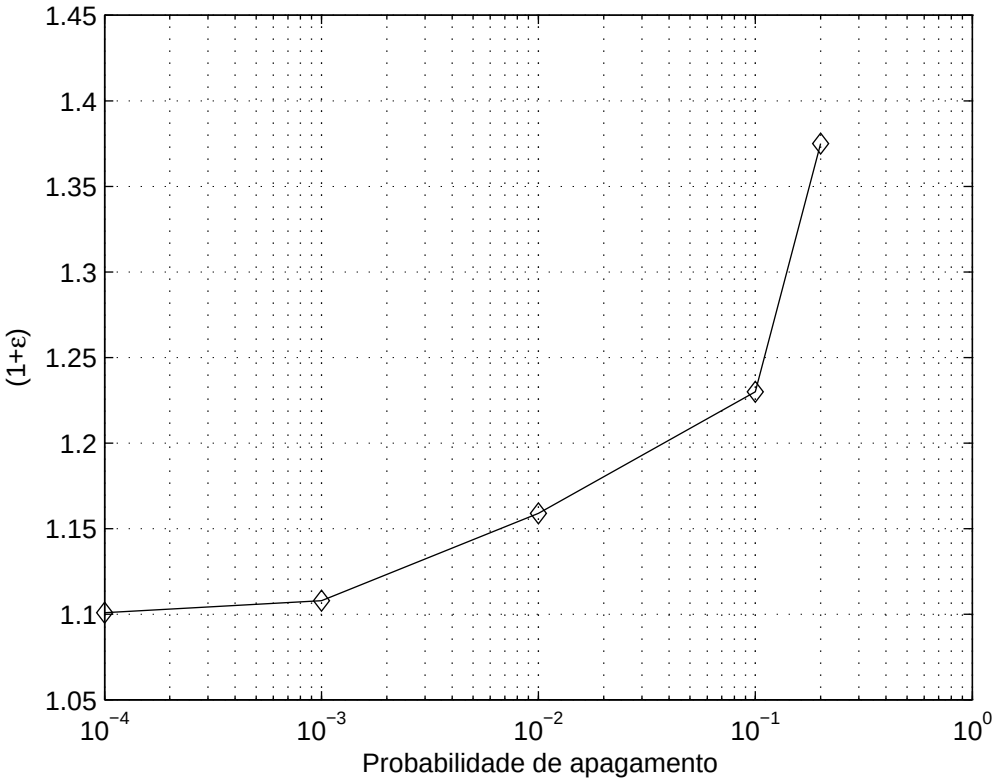


Figura 2.5: Desempenho do código LT em canais com apagamento.

tamente com o seu cabeçalho (o qual contém as informações sobre como o símbolo codificado foi formado). Desta forma, a probabilidade de um apagamento ser recebido será:

$$Pr[apagamento] = 1 - (1 - p)^m.$$

Assim, pode-se facilmente calcular o desempenho dos códigos LT em um canal BSC, a partir de sua análise em canais com apagamento, pois um pacote corrompido será interpretado como um apagamento. É importante lembrar que para que os pacotes corrompidos possam ser identificados e posteriormente descartados, deve-se utilizar um bom código detector de erros como por exemplo um CRC-32 [17]. Caso além de detectar, queira-se corrigir erros nos pacotes, um bom código corretor de erros deve ser utilizado concomitantemente com o código LT, conforme exposto na seção anterior e ilustrado na Figura 2.4. Nos cálculos desta subseção considerou-se que o código corretor de erros apenas detecta, não corrigindo eventuais erros.

2.5 Comparação com códigos tradicionais para canais com apagamento

Tipicamente, implementações de codificação para canais com apagamento utilizam códigos de bloco com taxa previamente fixada. Pesquisas na área de redes de comunicação têm sugerido o uso de códigos Reed-Solomon (RS) [16] e códigos Tornado [25] para aplicações em que códigos para canais com apagamento são requeridos. Entretanto, existem severas limitações para implementar tais códigos em arquivos de tamanho prático, sendo assim, clara a vantagem da utilização de códigos LT para aplicações em que é necessário lidar com apagamentos. Dentre as vantagens dos códigos LT, quatro se destacam:

1. Tempo de codificação e decodificação da ordem de $k.\ln(k/\delta)$.
2. Capacidade de geração de um número praticamente ilimitado de símbolos de saída.
3. Geração de símbolos de saída em tempo real.
4. Não é necessário conhecer a estatística do canal *a priori*.

São descritas a seguir algumas limitações dos códigos Reed-Solomon e códigos Tornado para ficar então clara a vantagem que os códigos LT, apesar de suas limitações (taxa maior que a unidade, tempo de codificação e decodificação dependente do número de símbolos de entrada de forma não linear), apresentam sobre os mesmos .

2.5.1 Códigos Reed-Solomon

A primeira limitação para aplicações que utilizam códigos RS se dá devido ao fato do número de símbolos de entrada k e o número de símbolos de saída n serem limitados devido a considerações práticas como o tempo de codificação e decodificação. Uma grande vantagem dos códigos RS é que sendo a mensagem original formada por k símbolos de entrada, quaisquer k dos n símbolos de saída gerados são suficientes para recuperar a mensagem original. No entanto, para estes códigos, a ordem do corpo finito em uso limita o número de símbolos de saída que podem ser gerados. Em aplicações práticas, um valor típico para a ordem do corpo finito é 256, o que limita n a 255 símbolos. Pode-se usar alfabetos maiores mas isso pode fazer com que a complexidade computacional do processo de decodificação se torne alta demais [5, 8, 17].

Essa limitação no número de pacotes que podem ser gerados causa sérios problemas em algumas aplicações como, por exemplo, quando há perda de pacotes em redes de comunicação. Para um valor baixo de n, pode acontecer de não serem recebidos os k pacotes necessários à decodificação. Nesse

caso, o receptor vai ter que esperar um outro ciclo de transmissão, recebendo um alto número de pacotes duplicados, diminuindo a eficiência da técnica. Efeito similar ocorre se o usuário suspender temporariamente o *download* de um arquivo. Suspender o *download* pode resultar na recepção de um grande número de pacotes repetidos, dependendo de onde o processo foi interrompido; [4] e [26] tratam com detalhes muitos desses aspectos cujo tratamento mais profundo foge ao escopo dessa dissertação.

2.5.2 Códigos Tornado

Os códigos Tornado [25] se assemelham aos códigos LT no que concerne às regras de codificação e decodificação. Uma outra semelhança é que, assim como ocorre com os códigos LT, $(1 + \epsilon)k$ $(0 < \epsilon < 1)$ símbolos de saída devem ser recebidos para garantir o sucesso na decodificação[4]. Entretanto, algumas limitações não existentes nos códigos LT devem ser mencionadas.

A primeira grande limitação dos códigos Tornado é inerente à sua estrutura. Os códigos Tornado utilizam uma seqüência cascateada de grafos bipartite entre muitas camadas de símbolos, em que os símbolos de entrada encontram-se na primeira camada e os símbolos resultantes de cada passo intermediário do processo de codificação encontram-se em cada camada subseqüente. Assim, essa estrutura gráfica deve ser calculada em ambos, codificador e decodificador. É possível ainda calculá-la no codificador e comunicá-la ao decodificador. No entanto, este pré-processamento mostra-se muito dispendioso na prática, sendo ainda o tamanho dos grafos em questão proporcionais ao número de símbolos de saída n, necessitando-se de grafos diferentes para cada tamanho de arquivo. Com o uso dos códigos LT, o único pré-processamento necessário é o cálculo das distribuições de graus usadas na codificação, algo que pode ser feito de maneira rápida e fácil tanto no codificador como no decodificador.

A segunda limitação diz respeito à quantidade de símbolos que podem ser gerados a partir de um conjunto de k símbolos de entrada. Como k e n devem ser fixados antes do processo de codificação, existe a impossibilidade de gerar símbolos em tempo real caso não sejam recebidos os $(1 + \epsilon)k$ símbolos necessários à decodificação. Já os códigos LT podem gerar símbolos em tempo real, podendo enviar quantos pacotes forem necessários para uma decodificação bem sucedida.

A terceira e talvez mais importante limitação reside na necessidade de se conhecer *a priori* a estatística do canal em uso. Isso porque, assim como os códigos RS, os códigos Tornado devem ter uma taxa pré-fixada. Assim, o que se faz na prática é usar a maior probabilidade de apagamento

[4] A grandeza $(1 + \epsilon)$ será referida daqui por diante como *overhead*.

possível para o canal em uso de forma a garantir o sucesso da decodificação. Tal abordagem leva obviamente a uma utilização subótima do canal.

A grande vantagem dos códigos Tornado é que os mesmos apresentam tempos de codificação e decodificação que variam linearmente com o tamanho do arquivo a ser codificado. No entanto, fazendo-se uso dos códigos Raptor [10] ou dos códigos em tempo real [11] pode-se unir as vantagens dos códigos LT (eliminação da necessidade do canal reverso, ausência de necessidade de conhecimento da estatística do canal, taxa variável facilmente ajustável ao longo da transmissão, complexidade dos processos de codificação e decodificação dependentes apenas do número de símbolos de entrada) a um tempo linear de codificação e decodificação.

CAPÍTULO 3

ANÁLISE ASSINTÓTICA DE CÓDIGOS LT

Probabilidade é grau de certeza e difere desta
como a parte difere do todo.

— James Bernoulli

Este capítulo dedica-se a apresentar a técnica de análise utilizada no cálculo da probabilidade de falha na decodificação de códigos LT quando o número de símbolos de entrada tende a infinito. Em primeiro lugar, descreve-se a técnica utilizada para logo após aplicá-la na avaliação da probabilidade acima referida.

3.1 A técnica de análise via avaliação de árvores E-OU

Esta seção explica de forma concisa a técnica de análise via árvores E-OU introduzida em [27], tendo em vista que esta é a principal ferramenta de análise a ser utilizada neste capítulo. Seja T_l uma árvore de profundidade $2l$; atribui-se a cada uma de suas folhas os valores 0 ou 1 de forma aleatória e independente. À raiz da árvore atribui-se profundidade 0, estando as folhas a uma profundidade $2l$. Cada nó com profundidade $0, 2, 4, ..., 2l-2$ é batizado como um nó "OU" (e a ele é atribuído o resultado do cálculo do "OU" booleano do valor de seus filhos), e cada nó com profundidade $1, 3, 5, ..., 2l-1$ é batizado como um nó "E" (e a ele é atribuído o resultado do cálculo do "E" booleano do valor de seus filhos).

Cada nó irá escolher independentemente quantos filhos terá. Para os nós "OU", a escolha é feita seguindo uma distribuição de probabilidades $(\alpha_0, \alpha_1, ..., \alpha_A)$ em que α_i é a probabilidade do nó

"OU" possuir i filhos. Já para os nós "E" a escolha é feita seguindo uma distribuição de probabilidades $(\nu_0, \nu_1, ..., \nu_B)$ em que ν_j é a probabilidade do nó "E" possuir j filhos. Aos nós "OU" sem filhos será atribuído o valor lógico 0, já aos nós "E" sem filhos será atribuído o valor lógico 1. Por fim, a cada folha serão associados os valores 0 ou 1 independentemente, sendo y_0 a probabilidade de uma folha assumir o valor 0. Como os nós de profundidade $2l$ serão folhas, não terão filhos. A Figura 3.1 ilustra uma árvore E-OU; perceba que os nós "OU" assumem o valor do resultado da operação "OU" booleana aplicada sobre o valor de seus filhos, de forma análoga, os nós "E" assumem o valor do resultado da operação "E" booleana aplicada sobre o valor de seus filhos.

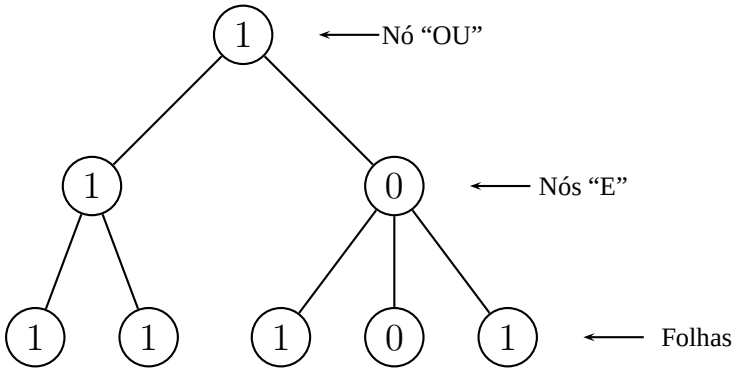


Figura 3.1: Árvore E-OU T_l com $l = 1$.

Tratando a árvore como um circuito booleano, o interesse da presente análise reside em calcular a probabilidade de que a raiz da árvore assuma o valor 0 (y_l). O seguinte lema exposto em [27] nos municia com um método para calcular y_l. A prova deste lema é bem direta, basta considerar que os nós "OU" de profundidade igual a 2 em T_l são as raízes de árvores E-OU independentes, T_{l-1}. Desta forma, a probabilidade y_l pode ser calculada como função de y_{l-1}, a probabilidade de que a raiz de uma árvore E-OU T_{l-1} assuma o valor 0.

Lema 3.1 *(Lema da árvore E-OU) A probabilidade y_l de que a raiz de uma árvore E-OU T_l assuma o valor 0 é $y_l = f(y_{l-1})$, em que y_{l-1} é a probabilidade de que a raiz de uma árvore E-OU T_{l-1} assuma o valor 0, e*

$$f(x) = \alpha(1 - \nu(1 - x)), \ \ em\ que$$
$$\alpha(x) = \sum_{i=0}^{A} \alpha_i x^i \ \ e \ \ \nu(x) = \sum_{i=0}^{B} \nu_i x^i.$$

A seguir expõe-se como essa técnica será utilizada na avaliação da probabilidade de falha na

decodificação de códigos LT de tamanho infinito, ou seja, com o número de símbolos de entrada tendendo a infinito.

3.2 Probabilidade de falha na decodificação de códigos LT de tamanho infinito

A ferramenta introduzida na seção anterior tem sido por excelência a mais utilizada no cálculo da probabilidade de falha na decodificação de códigos sem-taxa quando o número de símbolos de entrada tende a infinito (análise assintótica). Abordagens que utilizam tal ferramenta podem ser encontradas em [10], [11] e [28]; aqui, tal ferramenta será utilizada na análise de códigos LT.

Para iniciar o estudo, alguns conceitos devem ser relembrados. Seja $\mathcal{G}$ um grafo bipartite com k nós no lado esquerdo, n nós no lado direito e $\mathcal{E}$ ramos no total entre os lados esquerdo e direito. Será associado um símbolo de entrada a cada nó do lado esquerdo (nós de símbolo) e um símbolo de saída a cada nó do lado direito (nós de verificação). Seguindo a abordagem exposta em [10] e [11], pode-se modelar o processo de decodificação da seguinte maneira: a cada iteração do algoritmo de decodificação, mensagens (0 ou 1) são enviadas através dos ramos dos nós de verificação para os nós de símbolo e vice-versa.

Um nó de símbolo envia 0 para um nó de verificação adjacente, se e somente se, seu valor não foi recuperado ainda; da mesma forma, um nó de verificação envia 0 para um nó de símbolo adjacente, se e somente se, ele não puder recuperar tal nó de símbolo. Não é difícil verificar que os nós de símbolo podem ser vistos como nós “OU” (visto que para estes assumirem o valor 1 basta que um de seus filhos assuma o valor 1, isto é, para que um nó de símbolo seja recuperado basta que um dos nós de verificação envie uma mensagem reportando que pode recuperá-lo) e de forma análoga os nós de verificação podem ser vistos como nós “E”.

3.2.1 Distribuição de graus dos nós de símbolo

O primeiro passo para a aplicação da técnica de análise baseada em árvores E-OU consiste na demonstração de que a distribuição de graus de um nó de símbolo segue uma distribuição de Poisson com média constante. A presente análise segue a abordagem mostrada em [11]. A demonstração inicia-se com a descrição do processo de formação dos nós de verificação. Considere que serão formados βk nós de verificação, com $\beta > 1$.

Em primeiro lugar, cada um dos βk nós de verificação escolhe seu grau. Feito isto, os nós de símbolo são conectados aleatória e uniformemente através de ramos aos nós de verificação. A

primeira etapa determina o número total de ramos $\mathcal{E}$, o qual é calculado (tratando-se aqui de valores esperados) como sendo $\mathcal{E} = \beta\gamma k$, em que $\gamma = \sum_{i=1}^{k} i\Omega_i$ é o grau médio dos nós de verificação. Uma vez determinado o número de ramos, a segunda etapa pode ser encarada como o lançamento de $\beta\gamma k$ bolas (os ramos) em n urnas (os nós de verificação). Assim, a probabilidade λ_d de um nó de símbolo ter grau igual a d é dada por:

$$\lambda_d = \binom{\beta\gamma k}{d} p^d (1-p)^{\beta\gamma k - d}, \tag{3.1}$$

onde $p = p(k) = \frac{1}{k}$. Assintoticamente ($k \to \infty$), pode-se aproximar a distribuição de probabilidade (3.1) por uma distribuição de Poisson se as seguintes condições forem satisfeitas [21]:

1. $p(k) = o(1)$[5].
2. $\beta\gamma k p$ é uma constante.

Satisfazendo essas condições, λ_d pode ser aproximada por:

$$\lambda_d = \frac{e^{-\gamma\beta}(\gamma\beta)^d}{d!}.$$

No que segue, será admitido que as condições 1 e 2 são satisfeitas. A primeira delas é claramente satisfeita visto que $p(k) = \frac{1}{k}$. Já a condição 2 será atingida se $\beta\gamma$ for uma constante, o que é facilmente conseguido fazendo-se β (o *overhead*) e γ (o grau médio dos nós de símbolo) constantes. Assim, assintoticamente, ambos os nós da esquerda e da direita possuem graus limitados por constantes (para os nós da direita tal limite advém das regras de codificação).

3.2.2 Cálculo da probabilidade de falha na decodificação

Feitas as considerações preliminares, a probabilidade de um nó de símbolo ser recuperado pode ser calculada utilizando a análise via árvores E-OU em um subgrafo $\mathcal{G}_l$ construído da seguinte forma:

1. Escolha um ramo (v, w) de $\mathcal{G}$ aleatória e uniformemente. O nó v será a raiz de $\mathcal{G}_l$.
2. Remova (v, w) de $\mathcal{G}$.
3. $\mathcal{G}_l$ consistirá de todos os vértices de $\mathcal{G}$ que são alcançáveis percorrendo no máximo $2l$ ramos a partir de v e todos os ramos de $\mathcal{G}$ que conectam quaisquer pares dos referidos vértices.

A Figura 3.2 ilustra o processo de formação do subgrafo $\mathcal{G}_l$, para $l = 1$, a partir de um grafo $\mathcal{G}$. Escolhe-se um ramo de $\mathcal{G}$ de forma aleatória e uniforme **a)**. O ramo escolhido, indicado pela

[5]Diz-se que uma função $f(x)$ é $o(g(x))$ se $\lim_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| = 0$.

linha tracejada, é excluído. Assim, $\mathcal{G}_l$ será formado por todos os vértices de $\mathcal{G}$ que são alcançáveis percorrendo no máximo $2l = 2$ ramos a partir do vértice **A** e todos os ramos de $\mathcal{G}$ que conectam quaisquer pares dos referidos vértices, conforme ilustrado em **b)** e **c)**.

Pode-se mostrar [27] que o subgrafo $\mathcal{G}_l$ é uma árvore quando k tende a infinito. Isso porque, conforme dito anteriormente, ambos os nós da esquerda e da direita possuem graus limitados por constantes. Assim, $\mathcal{G}_l$ não possui mais do que um número constante de vértices, logo, a probabilidade que $\mathcal{G}_l$ não seja uma árvore, ou seja, possua um ciclo, tende a 0 quando k tende a ∞.

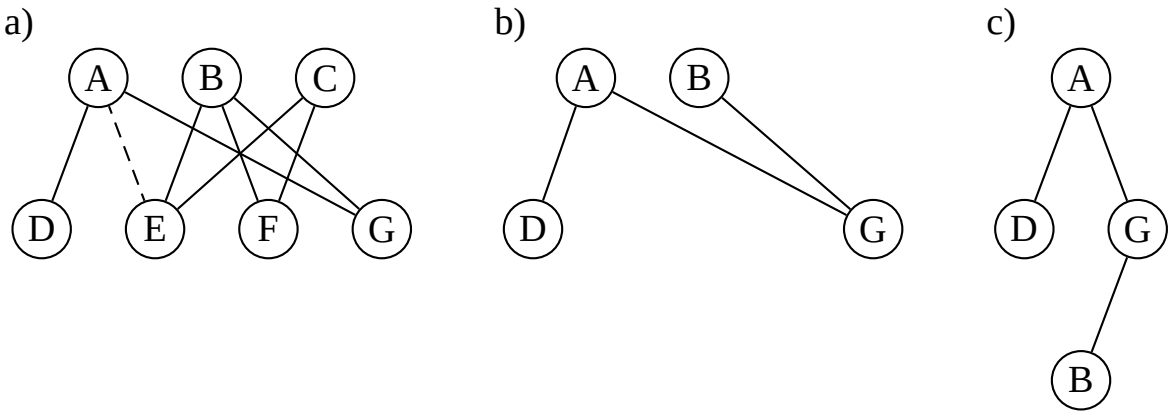


Figura 3.2: Processo de formação do subgrafo $\mathcal{G}_l$ para $l = 1$.

Estritamente falando, tal suposição só pode ser feita se o grau máximo dos nós de verificação for constante, ou seja, se o grau máximo dos símbolos de saída for constante. Utilizando a distribuição Sóliton Ideal ou Sóliton Robusta isso não ocorre, visto que para estas distribuições, o grau máximo dos símbolos de saída é k. No entanto, como a probabilidade de ocorrência de graus altos tende a zero quando $k \to \infty$, a análise baseada em árvores E-OU gera boas aproximações mesmo quando as distribuições Sóliton Ideal e Sóliton Robusta são utilizadas.

Desta forma, podemos usar o lema da árvore E-OU para calcular a probabilidade de um símbolo codificado não ser recuperado ao fim de l iterações, que nada mais é que a probabilidade y_l que a raiz de $\mathcal{G}_l$ assuma o valor 0; tal probabilidade será denotada por WER do inglês *word error rate*. A probabilidade de falha na decodificação de códigos LT pode ser facilmente obtida a partir da WER, basta perceber que o processo de decodificação falha se ao menos um símbolo se saída não for recuperado ao fim da decodificação. Considerando que o receptor disponha de n símbolos de saída, a probabilidade de falha de códigos LT será dada por:

$$\text{Pr[falha]} = 1 - (1 - WER)^n.$$

Usando a abordagem baseada em árvores E-OU, tem-se que ν_i é a probabilidade de que o ramo selecionado aleatoriamente na construção de $\mathcal{G}_l$ esteja ligado a um nó de verificação com $(i+1)$ filhos. Esta é a probabilidade de que o ramo esteja ligado a um nó de verificação de grau $(i+1)$. Logo:

$$\nu_i = \frac{(i+1)\Omega_{i+1}}{\Omega'(1)}, \tag{3.2}$$

o que implica em $\nu(x) = \frac{\Omega'(x)}{\Omega'(1)}$. Tal resultado é obtido a partir da observação de que o valor esperado da quantidade de nós de verificação de grau $(i+1)$ é $\beta k \Omega_{i+1}$. Assim, de um total de $\beta k \Omega'(1)$ ramos existentes, existem $(i+1)\beta k \Omega_{i+1}$ ramos ligados aos nós de verificação de grau $(i+1)$. Logo, a probabilidade de se escolher um ramo ligado a um nó de verificação de grau $(i+1)$ é dada por 3.2.

Analogamente, α_i é a probabilidade de que o nó de símbolo conectado ao ramo selecionado aleatoriamente na construção de $\mathcal{G}_l$ tenha grau $i+1$, o que pode ser facilmente calculado como sendo:

$$\alpha_i = \frac{(i+1)\lambda_{i+1}}{\beta\gamma} \tag{3.3}$$

o que, após substituições adequadas, resulta em $\alpha(x) = e^{\beta\gamma(x-1)}$. Resumindo tais resultados em um lema, tem-se:

Lema 3.2 *Considere um código LT com parâmetros $\Omega(x)$, k, e $\beta = (1+\epsilon)$. Seja y_l a probabilidade de que um nó de variável não seja recuperado após l iterações. Tem-se então:*

$$y_0 = 1,$$

e

$$y_l = \alpha(1 - \nu(1 - y_{l-1})), \quad l \geq 1,$$

onde

$$\nu(x) = \frac{\Omega'(x)}{\Omega'(1)} \;\; e \;\; \alpha(x) = e^{\beta\gamma(x-1)}.$$

Desta forma, pode-se calcular assintoticamente a probabilidade de falha na decodificação de um código LT, com distribuição de graus $\Omega(x)$. A Figura 3.3 ilustra o uso da técnica de análise assintótica para códigos LT para 20, 40, 80 e 100 iterações, utilizando a seguinte distribuição disponível em [10].

$$\Omega_1(x) = 0.007969x + 0.493570x^2 + 0.166220x^3 + 0.072646x^4$$

$$+0.082558x^5 + 0.056058x^8 + 0.037229x^9 + 0.055590x^{19} + 0.025023x^{65} + 0.0003135x^{66}.$$

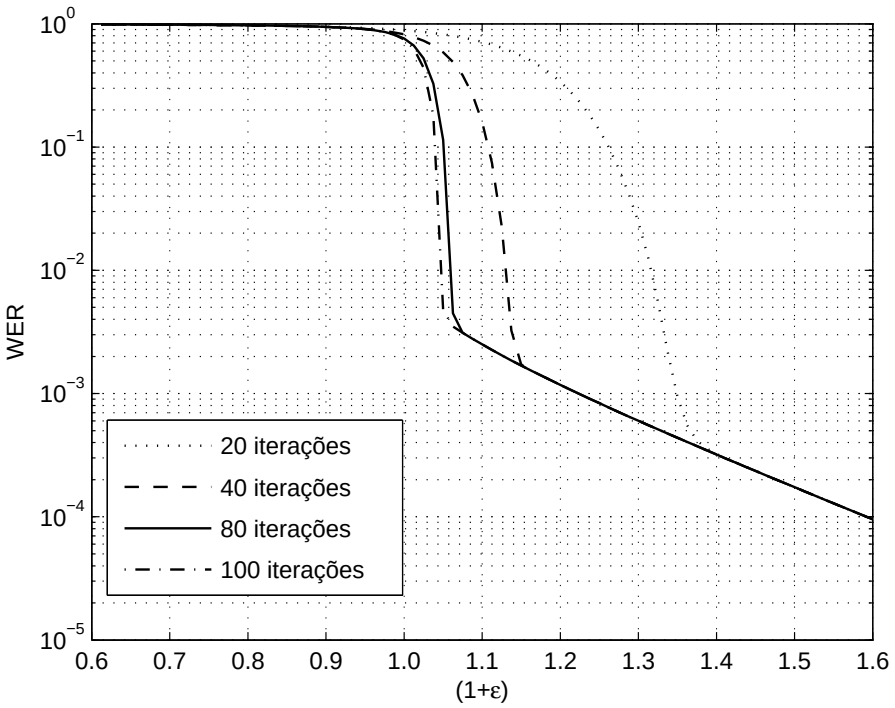


Figura 3.3: Probabilidade de um símbolo não ser recuperado em função do *overhead*.

É interessante notar que em conformidade com o esperado, ocorre a convergência das curvas à medida que o número de iterações cresce. Todas as curvas foram obtidas fazendo-se $k = 10000$.

Utilizando a distribuição $\Omega_1(x)$, foram realizadas simulações para um número de símbolos de entrada moderado, $k = 1000$. O resultado de tal simulação e o valor calculado pela análise assintótica encontram-se ilustrados na Figura 3.4. A metodologia empregada para se obter cada um dos pontos da simulação consistiu em gerar $(1 + \epsilon)k$ símbolos de saída e tentar recuperar os k símbolos de entrada a partir dos $(1 + \epsilon)k$ símbolos de saída gerados. Caso não fosse possível recuperar todos os símbolos de entrada, uma falha na decodificação era registrada. Para cada um dos pontos, simulações foram realizadas até a ocorrência de 100 falhas.

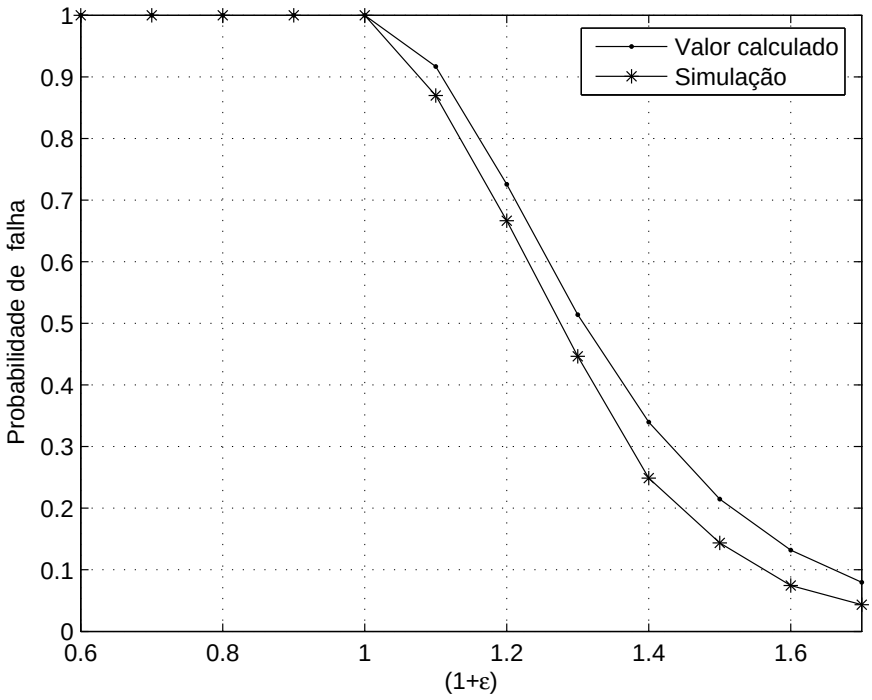


Figura 3.4: Probabilidade de falha em função do *overhead* para a distribuição $\Omega_1(x)$.

CAPÍTULO 4

PROJETANDO A DISTRIBUIÇÃO DE GRAUS

A sabedoria consiste em compreender que o tempo dedicado ao trabalho nunca é perdido.

— Ralph Emerson

NEste capítulo será analisado o processo de construção de uma boa distribuição de graus para os códigos LT. Ao se analisar o processo de codificação e decodificação destes códigos, fica patente a necessidade de se obter uma distribuição de probabilidade a qual atinja dois objetivos primordiais em relação ao conjunto de símbolos liberados mas ainda não processados: o *ripple*.

1. O *ripple* deve ser o menor possível ao longo do processo de decodificação para evitar uma repetição muito grande no referido processo, pois se um símbolo liberado já se encontra no *ripple*, nenhuma informação adicional foi obtida a partir da decodificação deste símbolo, desperdiçando esforço computacional. Assim, quanto menor o *ripple*, menor a probabilidade de haver liberação de símbolos os quais já se encontrem no mesmo.

2. O conjunto de símbolos liberados e não processados deve ser mantido em um tamanho grande o suficiente tal que não fique vazio antes da conclusão bem sucedida da decodificação, fato este que provocaria uma falha na decodificação (devido ao término da mesma sem que todos os símbolos de entrada tenham sido recuperados).

Conclui-se então que para conciliar estes dois objetivos conflitantes, uma característica desejável a uma boa distribuição seria processar os símbolos do *ripple* na mesma taxa em que outros são

adicionados ao mesmo (fato este que originou o nome Sóliton, pois um Sóliton é uma onda que equilibra perfeitamente a dispersão e a refração [29]). Além disso, uma boa distribuição de graus deve:

- ▷ Demandar, em média, a recepção da menor quantidade de símbolos de saída possível para garantir o sucesso da decodificação.
- ▷ Utilizar o menor número possível de operações para gerar um símbolo de saída. Isso é obtido mantendo-se baixo o grau médio dos símbolos de saída.

Uma distribuição de graus que atingiria os objetivos almejados seria aquela que liberasse exatamente um símbolo a cada iteração, pois assim o *ripple* seria mantido no menor tamanho possível (um símbolo) visto que o símbolo processado a cada iteração seria imediatamente substituído por outro, mantendo o *ripple* com apenas um símbolo ao longo de todo o processo de decodificação, tendo aquele uma alta probabilidade de nunca desaparecer antes da decodificação ser completada com sucesso. A seguir é descrita a distribuição Sóliton Ideal, a qual foi projetada para atingir tais objetivos.

4.1 A distribuição Sóliton Ideal

A presente dedução segue a notação proposta em [18]. Na primeira iteração $(l = 0)$, seja $h_0(d)$ o número de símbolos codificados de grau d. Assim sendo, o valor esperado do número de símbolos de grau d que tem seu grau reduzido para $d - 1$ na iteração seguinte é

$$h_0(d)\frac{d}{k},$$

em que k é o número de símbolos de entrada. Tal expressão nada mais é que o valor esperado de uma variável binomial, em que d/k é a probabilidade de um dos d vizinhos ser o símbolo presente no *ripple* a ser processado na iteração seguinte, reduzindo a $d - 1$ o grau de todos os símbolos que o tenham como vizinho. Na l-ésima iteração, quando l dos k símbolos de entrada tiverem sido recuperados e o número de pacotes de grau d for $h_l(d)$, o valor esperado do número de símbolos de grau d que terão seu grau reduzido para $d - 1$ é (lembre-se que a cada iteração um símbolo é processado):

$$h_l(d)\frac{d}{k-l}.$$

Assim, a fim de que o valor esperado do número de pacotes de grau 1 satisfaça a condição $h_l(1) = 1$ para todo $l \in \{0, ..., k-1\}$, devemos ter inicialmente $h_0(1) = 1$ e $h_0(2) = k/2$; mais genericamente, $h_l(2) = (k - l)/2$.

Para deduzir a distribuição Sóliton Ideal deve-se seguir o seguinte raciocínio: na segunda iteração $(l = 1)$ qual o número de símbolos de grau 2? É o número de símbolos de grau 3 que tiveram seu grau reduzido para 2, mais o número de símbolos de grau 2 que não foram reduzidos a grau 1, ou seja:

$$h_1(2) = \frac{h_0(3).3}{k} + \left(h_0(2) - \frac{h_0(2).2}{k} \right).$$

Da mesma forma, na terceira iteração $(l = 2)$ qual o número de símbolos de grau 2? É o número de símbolos de grau 3 que tiveram seu grau reduzido para 2, mais o número de símbolos de grau 2 que não foram reduzidos a grau 1 na iteração anterior, ou seja:

$$h_2(2) = \frac{h_1(3).3}{k-1} + \left(h_1(2) - \frac{h_1(2).2}{k-1} \right).$$

Assim, mais geralmente tem-se para $l > 0$:

$$h_l(d) = \frac{h_{l-1}(d+1).(d+1)}{k-(l-1)} + \left(h_{l-1}(d) - \frac{h_{l-1}(d).d}{k-(l-1)} \right). \tag{4.1}$$

Sendo o interesse da presente dedução encontrar uma formula geral para $h_0(d)$, é necessário eliminar na Equação 4.1 a dependência entre termos de iterações distintas, ou seja, eliminar a recursividade. Desta forma, após algumas manipulações, pode-se chegar à seguinte equação:

$$h_0(d) = \frac{2.(-1)^{d-2}}{d!}.h_0(2) + \sum_{j=1}^{d-3} \left[(-1)^{d-2-j}.\frac{2(d-2)(k-j-2)!k!}{(k-d)!(k-d)!d!}.h_j(2) \right] + \\ + \frac{2.k!}{(k-d+2)!d!}.h_{d-2}(2),$$

sabendo-se que $h_l(2) = (k-l)/2$ e simplificando a expressão acima chega-se a:

$$h_0(d) = (-1)^{d-2}.\frac{k(k-2)!}{(k-d)!d!} + (-1)^{d-2}.\frac{k!(d-2)}{(k-d)!d!}.\sum_{j=1}^{d-3} \left[\frac{(-1)^j}{k-j-1} \right] + \frac{k!}{(k-d+1)!d!}. \tag{4.2}$$

Utilizando tal expressão encontramos:

$$h_0(3) = \frac{k}{2.3},$$
$$h_0(4) = \frac{k}{3.4},$$
$$h_0(5) = \frac{k}{4.5},$$

confirmando o que pode ser obtido por induçao em (4.2), ou seja, para $d > 1$:

$$h_0(d) = \frac{k}{d(d-1)}.$$

No entanto, é notório que como o número de símbolos de grau d na primeira iteração $h_0(d)$ é o valor esperado de uma variável binomial, temos que $n.\rho_d = h_0(d)$, em que ρ_d é a probabilidade de um símbolo codificado ter grau d. Logo, fazendo $n = k$:

$$\rho_1 = 1/k \quad \text{visto que} \quad h_0(1) = 1,$$

$$\rho_d = \frac{1}{d(d-1)} \quad \text{para} \quad d = 2, ..., k,$$

que nada mais é que a distribuição Sóliton Ideal desenvolvida por Luby em [6]. Nas Figuras 4.1 e 4.2 tem-se uma ilustração do processo de decodificação. As setas indicam o número de símbolos de graus $d = 1, 2, 3$ e 4 que são reduzidos a um grau $d-1$ na primeira iteração (Figura 4.1) e na segunda iteração (Figura 4.2). Perceba que em ambas as iterações, apenas um símbolo de grau 2 é reduzido a grau 1, conforme requerido pela distribuição Sóliton Ideal. Os blocos sombreados representam os símbolos de grau d que são reduzidos a grau $d - 1$ a cada iteração.

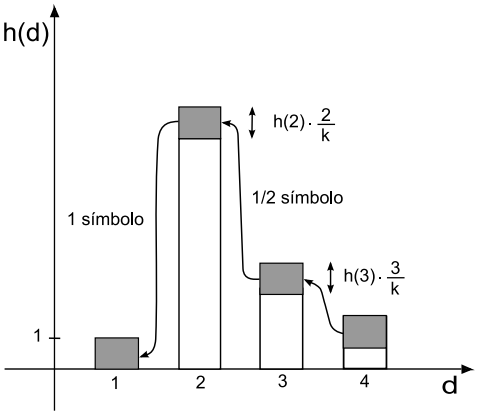


Figura 4.1: Distribuição Sóliton Ideal: primeira iteração.

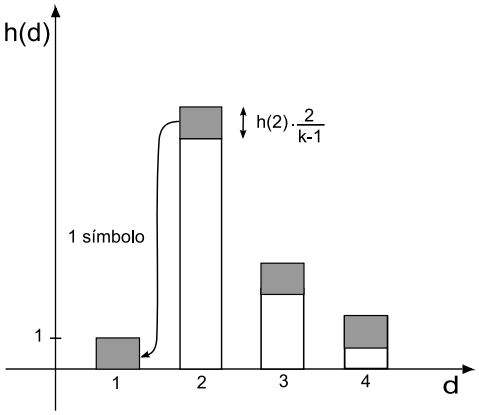


Figura 4.2: Distribuição Sóliton Ideal: segunda iteração.

A distribuição Sóliton Ideal, apesar de teoricamente apresentar, como seu nome sugere, um comportamento ideal, na implementação de sistemas ela mostra-se pouco útil, visto que a menor variância no valor esperado do número de símbolos no *ripple* (um símbolo) ao longo do processo de decodificação acarreta um esvaziamento do mesmo, o que leva a uma falha na decodificação. No entanto, a distribuição Sóliton Ideal pode ser modificada de forma que haja uma alta probabilidade de que o *ripple* nunca se esvazie antes do processo de decodificação ter sido completado. Essa modificação leva a uma outra distribuição de probabilidade, a distribuição Sóliton Robusta. A Figura 4.5 ilustra o desempenho da distribuição Sóliton Ideal, ou seja, a probabilidade de falha na decodificação após $(1 + \epsilon)k$ símbolos de saída terem sido recebidos (PF) e a probabilidade de um símbolo de entrada

não ser recuperado após a decodificação de $(1+\epsilon)k$ símbolos de saída (WER). As simulações foram realizadas com $k = 10000$ símbolos de entrada usando a análise assintótica exposta no Capítulo 3.

4.2 A distribuição Sóliton Robusta

O grande problema com a distribuição Sóliton Ideal é que o tamanho esperado do *ripple* é de apenas um símbolo. Qualquer variação no tamanho deste pode fazer com que o mesmo desapareça e o processo de decodificação falhe. A distribuição Sóliton Robusta procura corrigir essa falha, fazendo com que o tamanho do *ripple* seja grande o suficiente para que a cada passo do processo a probabilidade do mesmo desaparecer seja bem pequena. Ao mesmo tempo, a distribuição supramencionada procura obter um tamanho esperado para o *ripple* pequeno o bastante para minimizar a redundância advinda de símbolos codificados liberados cobrindo símbolos de entrada que já se encontrem no referido *ripple*.

O projeto da distribuição de que trata essa subseção visa manter o valor esperado para o tamanho do *ripple* (R) em torno de $\ln(k/\delta)\sqrt{k}$ ao longo do processo de decodificação, sendo δ a probabilidade de falha no processo. A escolha desse valor se dá devido à observação de que o processo de decodificação (crescimento e diminuição do *ripple*) se assemelha a um passeio aleatório; e em um passeio aleatório de comprimento k, a probabilidade de que o mesmo se desvie de sua média por um valor maior que $\ln(k/\delta)\sqrt{k}$ é no máximo δ [6]. Assim, δ é o limite superior da probabilidade de erro na decodificação.

Repetindo a análise utilizada na dedução da distribuição Sóliton Ideal, porém com o valor esperado do número de símbolos de grau 1 sendo agora $h_l(1) = 1+R$ para todo l, ou seja, $h_0(1) = 1+R$. Perceba entretanto, que ao contrário da distribuição Sóliton Ideal, o valor esperado do número de símbolos codificados de grau 2 que devem ter o grau reduzido para a unidade não mais vale 1, isso porque dentre os R símbolos adicionais pode haver repetição. Assim, existe uma probabilidade de R/k de que o símbolo processado na iteração encontre-se entre os R símbolos já presentes no *ripple*. Logo, o valor esperado do número de símbolos de grau 2 que devem ter seu grau reduzido para 1 vale $1 + R/k$, ou seja,

$$\frac{h_0(2).2}{k} = 1 + \frac{R}{k},$$

o que implica,

$$h_0(2) = \frac{k}{2} + \frac{R}{2}.$$

Na l-ésima iteração, quando l símbolos tiverem sido recuperados, tem-se:

$$\frac{h_l(2)}{k-l} = \frac{1}{2} + \frac{R}{2(k-l)}.$$

Usando o mesmo procedimento que culminou na obtenção da Equação 4.2 na dedução da distribuição Sóliton Ideal, obtém-se

$$h_0(d) = \frac{k}{d(d-1)} + \frac{R}{d},$$

para $1 < d \leq k$. Como o número de símbolos de grau d na primeira iteração, $h_0(d)$, é o valor esperado de uma variável binomial, temos (fazendo $n = k$) que $k.\mu_d^{'} = h_0(d)$ onde $\mu_d^{'}$ é a probabilidade de um símbolo ter grau d, logo:

$$\mu_d^{'}\frac{1}{d(d-1)} + \frac{R}{k.d} = \rho_d + \tau_d.$$

Eis o porquê da escolha do incremento τ_d. Normalizando $\mu_d^{'}$ obtemos a distribuição Sóliton Robusta definida anteriormente. A razão para truncar τ_d para $d > k/R$ e substituí-lo pelo incremento $\tau_{n/R}$ é para assegurar que a complexidade da decodificação não cresça mais que $O(k.\ln(k))$.

Perceba que no início da decodificação τ_1 assegura que o valor esperado do *ripple* seja $1 + R$ inicialmente. O incremento final $\tau_{k/R}$ assegura que todos os símbolos de entrada não cobertos, quando o número destes for igual ao tamanho do *ripple* (R), sejam todos cobertos. Em outras palavras, o incremento final $\tau_{k/R}$ assegura que todos os símbolos de entrada sejam selecionados, ao menos uma vez, como vizinhos de um símbolo codificado. Isto é similar ao processo de liberar simultaneamente $R.\ln(R/\delta)$ bolas (símbolos codificados) para cobrir R urnas (símbolos de entrada)[6].

Nas Figuras 4.3 e 4.4 tem-se uma ilustração do processo de decodificação usando a distribuição Sóliton Robusta. A interpretação é a mesma das figuras da Sóliton Ideal, as setas na Figura 4.4 indicam os símbolos de graus $d = 2, 3$ e 4 que são reduzidos a grau $d - 1$ na segunda iteração. Os blocos escuros representam 1 símbolo, já os mais claros representam símbolos liberados os quais já se encontram no *ripple*, não acrescentando assim nenhuma informação ao processo de decodificação.

4.2.1 Análise da distribuição Sóliton Robusta

São expostos aqui alguns resultados básicos envolvendo a distribuição Sóliton Robusta. Calcula-se aqui o número de símbolos codificados e o grau médio dos símbolos código, provando assim as expressões já expostas anteriormente.

[6] A análise do processo clássico de lançar bolas em urnas mostra que são necessários $k.\ln(k/\delta)$ bolas para que todas as k urnas sejam cobertas, ou seja, possuam no mínimo uma bola com probabilidade $1 - \delta$.

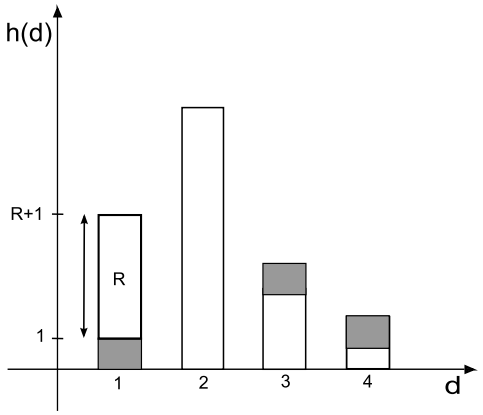


Figura 4.3: Distribuição Sóliton Robusta: primeira iteração.

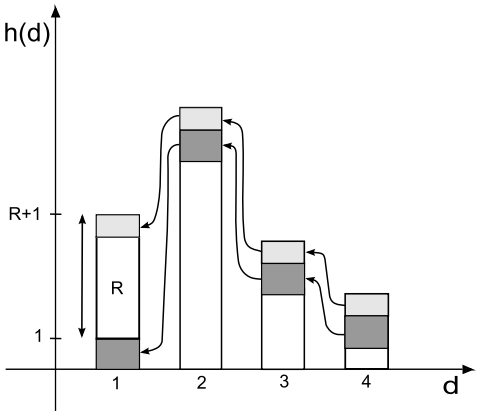


Figura 4.4: Distribuição Sóliton Robusta: segunda iteração.

Teorema 4.1 *O número de símbolos codificados é* $n = k + O(\sqrt{k}.\ln^2(k/\delta))$.

Prova:

$$n = k\beta = k\left(\sum_i \rho_i + \tau_i\right)$$

$$= k + \sum_{i=1}^{k/R-1} \frac{R}{i} + R.\ln(R/\delta)$$

$$\leq k + R.H(k/R) + R.\ln(R/\delta).$$

Como $H(k) \approx ln(k)$ a última equação equivale a,

$$\approx k + R.\ln(k/R) + R.\ln(R/\delta)$$

$$= k + \sqrt{k}.\ln^2(k/\delta)$$

logo,

$$n = k + O(\sqrt{k}.\ln^2(k/\delta)).$$

■

A "função" $O(\cdot)$ é uma notação freqüentemente utilizada para medir a complexidade computacional de um determinado algoritmo. Formalmente define-se $O(\cdot)$ da seguinte maneira:

Definição 4.1 *Sejam* $f(x)$ *e* $g(x)$ *duas funções definidas em algum subconjunto dos números reais. Diz-se que* $f(x)$ *é* $O(g(x))$ *quando* $x \to \infty$ *se e somente se:*

$$\exists\, x_0 > 0, \exists\, M > 0 \textit{ tal que } |f(x)| \leq\ M|g(x)| \textit{ para } x > x_0$$

.

Uma outra função utilizada no teorema anterior é a função $H(k)$ que nada mais é do que a série harmônica truncada $H(k) = \sum_{j=1}^{k} \frac{1}{j}$. A aproximação $H(k) \approx ln(k)$, assim como uma abordagem detalhada sobre análise de complexidade computacional podem ser encontradas em [30].

Teorema 4.2 *O grau médio de um símbolo código é* $\gamma = O(\ln(k/\delta))$[7].

Prova:

$$\gamma = \frac{\sum_i .(\rho_i + \tau_i)}{\beta}$$

$$\leq \sum_i i.(\rho_i + \tau_i)$$

$$= \sum_{i=2}^{k+1} \frac{1}{i-1} + \sum_{i=1}^{k/R-1} \frac{R}{k} + \ln(R/\delta)$$

$$\leq H(k) + 1 + \ln(R/\delta).$$

Como $H(k) \approx \ln(k)$ vem,

$$\gamma = O(\ln(k/\delta)).$$

■

Em [6] Luby prova que existe um c tal que a probabilidade de erro na decodificação de um código LT usando a distribuição Sóliton Robusta é limitada superiormente por δ. No entanto, tal análise é tão pessimista, que mesmo usando uma distribuição com altos valores de δ ($\delta = 0.5$ por exemplo), obtém-se códigos LT com bom desempenho. Assim sendo, evitaremos uma descrição do processo tedioso que leva à obtenção de um cota tão "frouxa". Os detalhes dessa derivação encontram-se no referido artigo.

Na Figura 4.5 encontra-se ilustrado o desempenho da distribuição Sóliton Robusta com parâmetros c e δ iguais a 0.01 e 0.1 respectivamente. Encontram-se ilustradas a probabilidade de falha da decodificação (PF) quando $(1+\epsilon)k$ símbolos de saída são recebidos e a probabilidade de um símbolo de entrada não ser recuperado (WER) ao fim da decodificação de $(1 + \epsilon)k$ símbolos de saída. As simulações mais uma vez foram implementadas para $k = 10000$.

4.3 Projeto de distribuições usando programação linear

As distribuições de graus expostas acima foram desenvolvidas baseadas nas características necessárias ao *ripple* para garantir o sucesso do processo de decodificação. Em [25], os autores sugerem um

[7] O uso da igualdade é um abuso de notação comumente utilizado que significa: γ é da ordem de $\ln(k/\delta)$

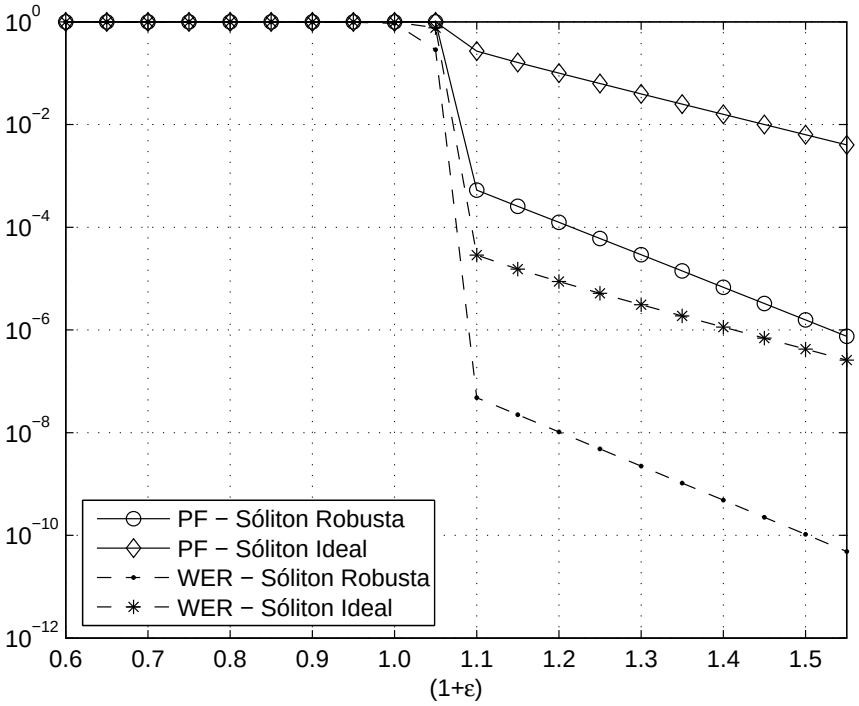


Figura 4.5: Desempenho das distribuições Sóliton Robusta com $c = 0.01$ e $\delta = 0.1$ e Sóliton Ideal.

processo baseado em programação linear de modo a desenvolver distribuições de graus que proporcionem uma alta probabilidade de sucesso na decodificação dos códigos Tornado. Posteriormente, Shokrollahi em [10] fez uma abordagem do referido processo para o desenvolvimento de códigos Raptor [10]; explanar tal abordagem e aplicá-la aos códigos LT é o objetivo dessa seção.

O projeto usando programação linear baseia-se na análise de processos aleatórios através de árvores E-OU. Seja p_l a probabilidade de um ramo no grafo de decodificação enviar o valor 1 de um nó de saída para um nó de entrada na l-ésima iteração. Usando o lema da árvore E-OU obtém-se

$$p_{l+1} = \nu(1 - \alpha(1 - p_l)), \quad l \geq 1,$$

onde

$$\nu(x) = \frac{\Omega'(x)}{\Omega'(1)} \ \ e \ \ \alpha(x) = e^{\beta\gamma(x-1)},$$

sendo $\nu(x)$ e $\alpha(x)$ as distribuições de graus dos nós de saída e entrada, respectivamente. Seja u_l a probabilidade de um símbolo de entrada ser recuperado na iteração l. Convém lembrar que um símbolo de entrada é recuperado, se e somente se, estiver ligado a um ramo que envia o valor 1 de um nó de saída para um nó de entrada. Assim, a probabilidade de um símbolo de grau d ser recuperado pode ser facilmente calculada como sendo $1 - (1 - p_l)^d$. Logo, supondo que o símbolo de entrada

tenha grau d, $\alpha_i = 0$ para todo $i \neq d$, ou seja, $(1-p_l)^d = \alpha(1-p_l)$. Então, a probabilidade de um símbolo de entrada ser recuperado na l-ésima iteração do algoritmo é $u_l = 1-\alpha(1-p_l) = 1-e^{-\beta\gamma p_l}$. Escrevendo em forma de recursão em u tem-se:

$$u_{l+1} = 1 - e^{-\beta\gamma\nu(u_l)}.$$

A recursão acima mostra que, se uma fração esperada de x símbolos de entrada foi recuperada em alguma iteração l, na próxima iteração $l+1$ essa fração aumenta para $1 - e^{-\beta\gamma\nu(x)}$. Logo, a fração esperada de símbolos no ripple em l será $1 - x - e^{-\beta\gamma\nu(x)}$. Sendo o número de símbolos de saída igual a $\beta k = (1+\epsilon)k$, então $\beta\gamma\nu(x) = (1+\epsilon)\Omega'(x)$ sendo a fração esperada de símbolos no *ripple* dada por:

$$1 - x - e^{-\Omega'(x)(1+\epsilon)}.$$

Dito isto, é preciso projetar-se uma distribuição de graus para os símbolos de saída de forma a garantir que uma grande fração dos k símbolos de entrada sejam recuperados. Assume-se então, para fins de projeto, que a inserção e retirada de elementos do *ripple* é feita de forma independente com probabilidade $1/2$. Assim, o *ripple* deve ser maior por um fator r da raiz quadrada do número de símbolos de entrada ainda não recuperados, ou seja, maior que $r\sqrt{(1-x)k}$. Esse valor surge a partir da aplicação de uma cota de Chernoff [31] para a soma (X) de variáveis aleatórias i.i.d.[8] (X_i), onde $X = \sum_{i=1}^{n} X_i$ sendo $Pr[X_i = 1] = 1/2$ e $Pr[X_i = -1] = 1/2$. Seja R o valor esperado do *ripple*, o processo de entrada e saída de símbolos no mesmo pode ser visto como um passeio aleatório [21] de tamanho N, em que N é igual ao número de símbolos ainda não recuperados. Logo, conforme pode ser visto no Apêndice A,

$$\begin{aligned} Pr[X \leq a] &\leq e^{\frac{-a^2}{2N}} \\ = Pr[X \leq -R] &\leq e^{\frac{-R^2}{2(1-x)k}} \leq c', \end{aligned}$$

em que c' é uma constante que denota a máxima probabilidade de falha admissível. Assim, após algumas manipulações chega-se à condição a seguir para o tamanho do *ripple*:

$$R \geq r\sqrt{(1-x)k} \text{ para um dado } r = \sqrt{2\ln(1/c')}.$$

[8] independentes e identicamente distribuídas

Usando essa condição, o problema do projeto da distribuição dos símbolos de saída torna-se o seguinte: dados ϵ, δ e o número de símbolos de entrada k, encontre uma distribuição de graus tal que, para $x \in [0, 1-\delta]$,

$$k(1 - x - e^{-\Omega'(x)(1+\epsilon)}) \geq r\sqrt{(1-x)k},$$

em que $k(1 - x - e^{-\Omega'(x)(1+\epsilon)})$ é o tamanho do *ripple*. Reescrevendo a equação em termos de $\Omega'(x)$ tem-se,

$$\Omega'(x) \geq \frac{-\ln\left(1 - x - r\sqrt{\frac{1-x}{k}}\right)}{1+\epsilon},$$

para $x \in [0, 1-\delta]$. Note que para a desigualdade ter solução, δ deve ser maior que $r/\sqrt{k}$. Esse problema pode ser facilmente resolvido através de uma abordagem utilizando programação linear [32].

O primeiro passo consiste em escolher um conjunto de inteiros M o qual nós queremos que contenha os possíveis graus dos símbolos de saída. Em seguida, deve-se discretizar o intervalo $[0, 1-\delta]$ e fazer com que a desigualdade acima seja obedecida em tais pontos, obtendo-se assim um conjunto de desigualdades lineares em que as incógnitas são os coeficientes de $\Omega(x)$. Por fim, escolhe-se minimizar a função objetivo $\Omega'(1)$ (que é função dos coeficientes Ω_i para $i \in M$) a fim de obter uma distribuição de graus com o menor valor médio possível.

Na Tabela 4.1 encontram-se algumas distribuições obtidas a partir de simulações implementadas fazendo uso da abordagem acima exposta. Nela, $(1+\epsilon)$ é o *overhead*, γ é o grau médio dos símbolos de saída e γ_{SR} é o grau médio dos símbolos de saída da distribuição Sóliton Robusta para os mesmo valores de k. Ao contrário do valor adotado para a distribuição Sóliton Robusta, em todas as distribuições projetadas usando programação linear fez-se $\delta = 0.01$. O conjunto M escolhido nas presentes simulações foi o mesmo escolhido em [10].

A Figura 4.6 ilustra o desempenho das distribuições projetadas usando programação linear. É importante citar que as otimizações foram feitas incluindo as restrições de que para todo $i > 0$, $\Omega_i \geq 0$. Uma outra restrição foi o grau médio das distribuições, essa restrição é importante pois, apesar de se estar minimizando a função objetivo $\Omega'(1)$, graus médios muito baixos não resultam em boas distribuições no que diz respeito à probabilidade de falha na decodificação.

É possível observar na Figura 4.6 que, quanto maior o grau médio da distribuição, menor a probabilidade de falha para $(1+\epsilon) > 1$. Essa melhoria no desempenho, no entanto, implica uma maior complexidade computacional, visto que mais operações terão quer ser realizadas, em média,

Tabela 4.1: Distribuições de graus para vários valores de k.

k	1000	5000	10000	50000
Ω_1	0,069168	0,060131	0,056290	0,049966
Ω_2	0,385100	0,395150	0,398850	0,407210
Ω_3	0,209760	0,210130	0,209710	0,210000
Ω_4	0,095456	0,093559	0,092477	0,090704
Ω_5	0,057914	0,056956	0,056532	0,055473
Ω_8	0,042095	0,042853	0,043572	0,044244
Ω_9	0,041593	0,042165	0,042853	0,043530
Ω_{18}	0,028249	0,024327	0,022371	0,019771
Ω_{19}	0,027125	0,023004	0,020882	0,018070
Ω_{20}	0,026175	0,021916	0,019662	0,016678
Ω_{65}	0,003281	0,006324	0,008120	0,010137
Ω_{66}	0,005645	0,009709	0,012002	0,014481
Ω_{67}	0,008442	0,013771	0,016678	0,019735
ϵ	0.05	0.05	0.05	0.05
γ	5.55	6.15	6.50	6.85
γ_{SR}	10.35	13.28	14.58	17.57

para codificar e decodificar um símbolo. Todas as WER foram calculadas com 100 iterações. Faz-se mister lembrar WER denota a probabilidade de um símbolo de entrada não ser recuperado após $(1 + \epsilon)k$ símbolos de saída terem sido recebidos, acarretando assim, falha no processo de decodificação.

Analisando os gráficos de desempenho, pode-se questionar porque usar tal abordagem se a distribuição Sóliton Robusta apresenta menores probabilidades de falha quando $(1 + \epsilon) > 1.0$. A resposta é simples: usando programação linear pode-se controlar o grau médio dos símbolos de saída, algo impossível se a distribuição Sóliton Robusta for utilizada. Tal controle é extremamente importante quando se quer trocar uma baixa probabilidade de erro por uma baixa complexidade computacional nos processos de codificação e decodificação, isso porque quão maior o grau médio de uma distribuição, maior o número de operações a serem realizadas a fim de recuperar a mensagem original. Perceba na Tabela 4.1 que o melhor desempenho da distribuição Sóliton Robusta ocorre às custas de um alto grau médio, isso acarreta uma maior complexidade computacional aos processos de codificação e decodificação.

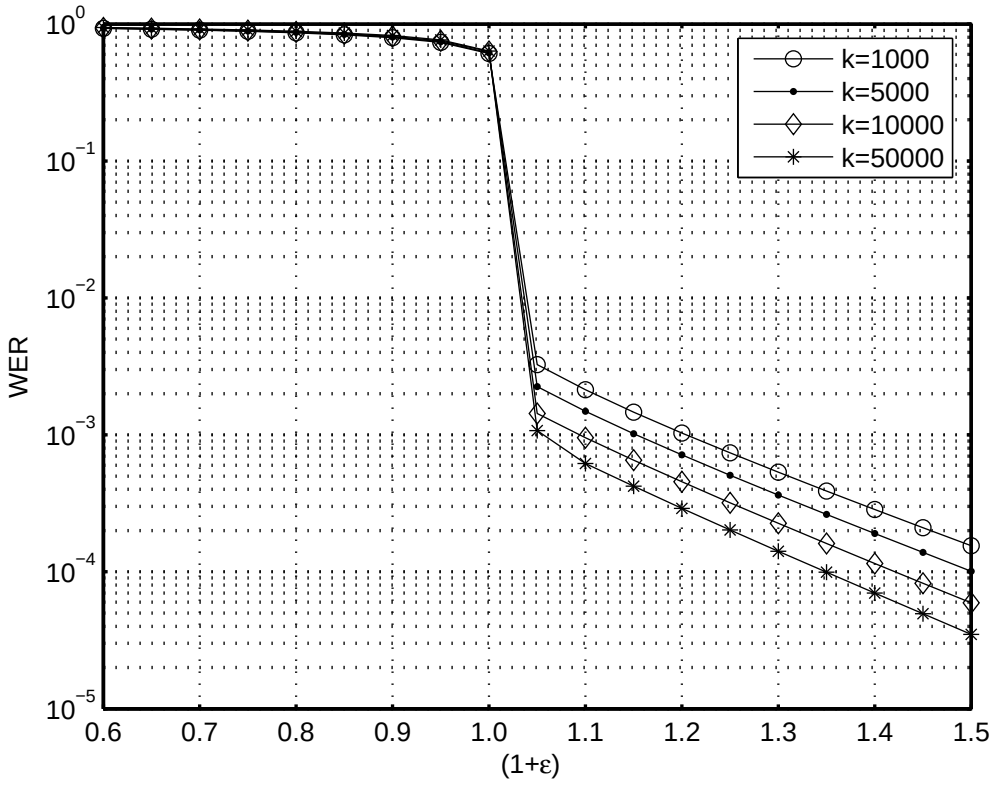


Figura 4.6: Desempenho das distribuições da Tabela 4.1.

CAPÍTULO 5

CONCLUSÃO, COMENTÁRIOS E SUGESTÕES

Quanto mais alto se sobe, mais longe é o horizonte.

— Vergílio Ferreira

APresenta-se a seguir a conclusão da presente dissertação juntamente com alguns comentários e sugestões de trabalhos futuros.

5.1 Conclusões finais e comentários

O objetivo primeiro dessa dissertação foi investigar a recém-criada classe de códigos para canais com apagamento conhecida como códigos LT. Apesar de já haver uma boa quantidade de informações na literatura acerca do objeto de estudo deste texto, verifica-se uma carência no que diz respeito a análise de desempenho desses códigos.

Primeiramente, foi introduzido o cenário em que os códigos LT encontram aplicação. Conceitos como canais com apagamento, códigos sem-taxa e fontes digitais foram expostos conjuntamente com possíveis aplicações dos códigos estudados. Feito isto, foram abordados no capítulo 2 os processos de codificação e decodificação para posteriormente ser analisado o desempenho dos códigos em tela em canais livres de ruído e canais com apagamento. Foi verificado que os códigos LT aproximam-se bem do conceito de fontes digitais enquanto mantêm tempos de codificação e decodificação extremamente pequenos em comparação com os códigos tradicionais para canais com apagamento.

No capítulo três, foram expostas técnicas de análise assintótica para códigos LT. Os resultados de

tal análise encontram-se ilustrados nesse capítulo. A explanação, baseada em conceitos elementares de teoria de probabilidades, do processo de obtenção das distribuições Sóliton Ideal e Sóliton Robusta oferece uma forma simples de entender a idéia por trás da concepção das mesmas. Exibe-se ainda o detalhamento de uma técnica baseada em programação linear para obtenção de boas distribuições de graus para códigos LT. Por fim, no capítulo 2 foi apresentada ainda uma pequena contribuição: um esquema de geração de cabeçalhos para códigos LT.

O esquema proposto mostra-se eficiente em dois aspectos: 1) necessita de um pequeno número de *bits* para sua construção; e 2) permite uma comunicação sem a necessidade de sincronismo entre o codificador e o decodificador. A primeira característica faz com que se minimize a quantidade de informação auxiliar, maximizando a quantidade de informação "útil" em cada símbolo codificado. De acordo com o que foi visto, a razão entre o número de *bits* necessários à representação da informação auxiliar e o número de *bits* contendo informação útil depende apenas do número de *bits* dos símbolos de entrada, assim, a representação torna-se mais eficiente à medida que o número de *bits* dos símbolos de entrada cresce. A segunda característica elimina um requisito que por vezes acrescenta complexidade aos sistemas de comunicação: a necessidade de sincronismo. O casamento dessas duas características gera um esquema de representação de cabeçalhos de fácil implementação e baixa complexidade conforme exposto na Seção 2.3.

Tal esquema mostrou-se muito eficiente no que diz respeito ao número de *bits* necessários a sua representação possibilitando ainda um processo de decodificação assíncrono, ou seja, o codificador e o decodificador não precisam estar sincronizados para obtenção das informações sobre quantos e quais são os vizinhos de cada símbolo codificado.

5.2 Sugestões de trabalhos futuros

Existe certamente uma vasta gama de trabalhos a serem realizados na área de códigos sem-taxa. Aplicar as técnicas expostas nessa dissertação ao estudo de outros códigos sem-taxa, como por exemplo códigos Raptor [10] e códigos em tempo real [11], certamente é uma sugestão de trabalho interessante.

A implementação de sistemas que utilizem códigos sem-taxa sobre os quais não existam patentes, como os códigos em tempo real (*online codes*) [11], conjuntamente com implementações eficientes (no que diz respeito ao tempo de decodificação) dos algoritmos de decodificação, também podem ser bons objetos de estudo. Por fim, aliar os conceitos presentes na dissertação em tela ao estudo de codificação para redes (*network coding*) [33] é uma linha de pesquisa que pode render bons frutos,

tendo em vista as enormes possibilidades de aplicação nos sistemas de comunicações modernos, sendo esse o estado da arte da pesquisa em redes de comunicação.

REFERÊNCIAS

[1] A. S. TANENBAUM, **Computer Networks**, 4ª ed. Prentice Hall, Agosto 2002.

[2] J. C. MOREIRA & P. G. FARRELL, **Essentials of Error-Control Coding**. John Wiley & Sons, Setembro 2006.

[3] A. M. MICHELSON & A. H. LEVESQUE, **Error-Control techniques for digital communication**, 1ª ed. John Wiley & Sons, 1985.

[4] M. LUBY, M. MITZENMACHER, & J.W.BYERS, A digital fountain approach to asynchronous reliable multicast, *IEEE J. on Selected Areas in Communications, Special Issue on Network Support for Multicast Communications*, 2002.

[5] J. W. BYERS, M. LUBY, M. MITZENMACHER, & A. REGE, A digital fountain approach to reliable distribution of bulk data, In: **Proc. ACM SIGCOMM**, Agosto 1998, p. 56–67.

[6] M. LUBY, LT codes, In: **Procedings of the ACM Symposium on Foundations of Computer Science**, 2002.

[7] D. J. C. MACKAY, Fountain codes, *IEE Proc.-Communication*, v. 152, n. 6, p. 1062–1068, Dezembro 2005.

[8] S. LIN & D. J. COSTELLO, **Error Control Coding**, 2ª ed. Prentice Hall, Abril 2004.

[9] P. MAYMOUNKOV & D. MAZIERES, Rateless codes and big downloads, In: **Proc. of the 2nd Int. Workshop Peer-to-Peer Systems (IPTPS)**, Fevereiro 2003.

[10] A.SHOKROLLAHI, Raptor codes, *IEEE International Symposium on Information Theory (ISIT) 2004*, Junho 2004.

[11] P. MAYMOUNKOV, Online codes, NYU, Relatório Técnico TR2003-883, Novembro 2002.

[12] A. W. ECKFORD & W. YU, Rateless slepian wolf codes, In: **Proc. Asilomar Conference on Signals, Systems and Computers**, Outubro - Novembro 2005, p. 1757–1761.

[13] B. LATHI, **Modern Digital and Analog Communication Systems**, 3ª ed. Oxford University Press, 1998.

[14] C. E. SHANNON, A mathematical theory of communications, *Bell System Technical Journal*, v. 27, p. 379–423 e 623–656, Julho e Outubro 1948.

[15] P. ELIAS, Coding for two noisy channels, *Information Theory, Third London Symposium*, p. 61–76, 1955.

[16] I. REED & G.SOLOMON, Polynomial codes over certain finite fields, *J. Soc. Indust. Appl. Math*, v. 8, p. 300–304, 1960.

[17] S. B. WICKER, **Error Control Systems for Digital Communication and Storage**. Prentice Hall, 1995.

[18] D. J. MACKAY, **Information Theory, Inference and Learning Algorithms**. Cambridge University Press, 2004.

[19] A. MAHANTI, D. EAGER, M. K.VERNON, & D. SUNDARAM-STUKEL, Scalable on-demand media streaming with packet loss recovery, *IEEE/ACM Transactions on Networking*, v. 11, n. 2, p. 195–209, Abril 2003.

[20] M. LUBY, Information additive code generator and decoder for communication systems, *U.S. Patent No. 6,307,487*, Out. 23, 2001.

[21] S. M. ROSS, **Introduction to Probability Models**, 8ª ed. Academic Press, 2003.

[22] ——, **Simulation**, 3ª ed. Academic Press, 2002.

[23] C. HARRELSON, L. IP, & W. WANG, Limited randomness LT codes, In: **The 41st Annual Allerton Conference on Communication, Control, and Computing**, 2003.

[24] A.SHOKROLLAHI, S.LASSEN, & M. LUBY, Multi-stage code generator and decoder for communication systems, *U.S. patent No 7,068,729*, Julho 27, 2006.

[25] M. LUBY, M. MITZENMACHER, & A. SHOKROLLAHI, Efficient erasure correcting codes, *IEEE Transactions on Information Theory*, v. 47, p. 569–584, Fevereiro 2001.

[26] M. MITZENMACHER, Digital fountains: A survey and look forward, *IEEE Information Theory Workshop*, p. 24–29, Outubro 2004.

[27] M. LUBY, M. MITZENMACHER, & A. SHOKROLLAHI, Analysis of random processes via and-or tree evaluation, *In Symposium on Discrete Algorithms*, 1998.

[28] N. RAHNAVARD, BADRI N. VELLAMBI R., & F. FEKRI, Rateless codes with unequal error protection, *não publicado*.

[29] J.SCOTT, Report on waves, *Report of the fourteenth meeting of the British Association for the Advancement of Science*, p. 311–390, Setembro 1844, plates XLVII-LVII.

[30] D. E. KNUTH, **The Art of Computer Programming, Volume 1: Fundamental Algorithms**, 3ª ed. Addison Wesley, 1997.

[31] M. MITZENMACHER & E. UPFAL, **Probability and Computing: Randomized Algorithms and Probabilistic Analysis**. Cambridge University Press, 2003.

[32] A. G. GLICKSMAN, **An introduction to linear programming and the theory of games**. John Wiley & Sons, 1963.

[33] C. FRAGOULI, J. L. BOUDEC, & J. WIDMER, Network coding: An instant primer, *SIGCOMM Comput. Commun. Rev.*, v. 36, n. 1, p. 63–68, 2006.

[34] R. M. TANNER, A recursive approach to low complexity codes, *IEEE Transactions on Information Theory*, v. 27, p. 533–547, 1981.

APÊNDICE A

COTAS DE CHERNOFF

As cotas de Chernoff são cotas de probabilidade derivadas aplicando-se a desigualdade de Markov às funções geradoras de momento de uma variável aleatória.

A.1 Funções geradoras de momentos

Definição A.1 *A função geradora de momentos de uma variável aleatória X é definida para qualquer valor real t como:*

$$M_X(t) = E[e^{tX}].$$

A.2 Cotas de Chernoff

Conforme dito anteriormente, a cota de Chernoff para uma variável aleatória é obtida aplicando-se a desigualdade de Markov a e^{tX} para algum valor apropriado de t. A partir da desigualdade de Markov pode-se obter para qualquer $t > 0$,

$$Pr(X \geq a) = Pr(e^{tX} \geq e^{ta}) \leq \frac{E[e^{tX}]}{e^{ta}}.$$

Em particular,

$$Pr(X \geq a) \leq \min_{t>0} \frac{E[e^{tX}]}{e^{ta}}.$$

Analogamente para $t < 0$,

$$Pr(X \leq a) = Pr(e^{tX} \geq e^{ta}) \leq \frac{E[e^{tX}]}{e^{ta}}.$$

Assim,

$$Pr(X \leq a) \leq \min_{t<0} \frac{E[e^{tX}]}{e^{ta}}.$$

É fácil notar que o valor de t que minimiza $E[e^{tX}]/e^{ta}$ gera a melhor cota possível, embora seja usual escolher o valor de t que gere uma forma conveniente para a equação da cota. Todas as cotas obtidas da maneira acima exposta são denominadas *Cotas de Chernoff*. Pode-se assim, obter uma cota apropriada para cada tipo de variável aleatória em estudo, é o que será feito a seguir.

Teorema A.1 *Seja $X_1, ..., X_n$ um conjunto de variáveis independentes onde $Pr[X_i = 1] = Pr[X_i = -1] = 1/2$. Então a seguinte cota de Chernoff se aplica:*

$$Pr[X \geq a] < e^{-a^2/2n}.$$

Prova: Aplicando a desigualdade de Markov obtém-se para qualquer $t > 0$,

$$Pr(X \geq a) = Pr(e^{tX} \geq e^{ta})$$
$$\leq \frac{E[e^{tX}]}{e^{ta}},$$

no entanto,

$$E[e^{tX}] = E[\prod_i e^{tX_i}]$$
$$= \prod_i E[e^{tX_i}] = \cosh^n(t)$$
$$\leq e^{t^2 n/2},$$

logo tem-se:

$$Pr(X \geq a) \leq e^{-ta+t^2 n/2},$$

escolhendo $t = a/n$ vem:

$$Pr(X \geq a) \leq e^{-a^2/2n}.$$

■

APÊNDICE B

GRAFOS

Ao longo deste documento, foi utilizada uma abordagem dos códigos LT baseada em grafos. O objetivo por trás de tal abordagem reside na facilidade de se calcular a probabilidade de falha na decodificação analisando a probabilidade de falha do algoritmo de decodificação baseado na análise de processos aleatórios utilizando árvores E-OU [27]. Encontram-se expostos aqui, os conceitos em teoria dos grafos utilizados ao longo da dissertação.

B.1 Grafos de Tanner

Definição B.1 *(Grafo) Um grafo $\mathcal{G}$ é uma estrutura que consiste em um conjunto de vértices (ou nós), denotado por $V = \{v_1, v_2, ...\}$, e um conjunto de ramos (disjunto de V), denotado por $E = \{e_1, e_2, ...\}$. Cada ramo e_k pertencente ao conjunto E está associado a um par (ordenado ou não-ordenado) (v_i, v_j) de vértices (não necessariamente distintos) do conjunto V. Os vértices v_i e v_j são chamados de vértices terminais de e_k. O grafo $\mathcal{G}$ é denotado por $\mathcal{G}$=(V,E).*

Grafos são geralmente mostrados com pontos representando os vértices e linhas representando os ramos. Os ramos de um grafo podem ser *direcionados* quando os vértices unidos por este são ordenados, ou *não-direcionados* caso contrário. Um grafo é dito simples se: 1) não existe nenhum ramo conectando um nó a si mesmo; 2) existe no máximo um ramo entre quaisquer pares de vértices; e 3) todos os ramos são não-direcionados. Em um grafo simples, diz-se que dois vértices são adjacentes se existe um ramo ligando os dois, ou seja, quaisquer v_i ,v_j pertencentes a **V** são ditos adjacentes se (v_i, v_j) pertence a **E**. Ao conjunto de todos os vértices adjacentes a um nó v_i dá-se o nome de vizinhança de v_i. Uma seqüência de vértices distintos, iniciada no vértice x e terminada no

vértice y, é chamada de *caminho* entre x e y se todos os vértices consecutivos nessa seqüência forem adjacentes. Um caminho fechado, ou ciclo, é todo aquele que se inicia e termina no mesmo nó.

Definição B.2 *(Grafo bipartite) Um grafo bipartite é um grafo cujo conjunto de vértices V é dividido em dois subconjuntos disjuntos, denotados por V_c e V_s onde nenhum par de vértices pertencente a um mesmo subconjunto é adjacente.*

Abaixo, nas Figuras B.1 e B.2, ilustra-se a definição de grafo bipartite. Os vértices brancos e pretos formam dois subconjuntos ligados entre si pelos ramos. Note que no segundo caso o grafo não é bipartite, pois vértices de um mesmo subconjunto possuem ramos ligando um ao outro.

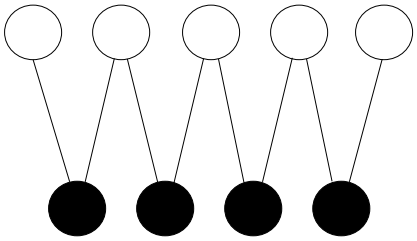


Figura B.1: Grafo bipartite.

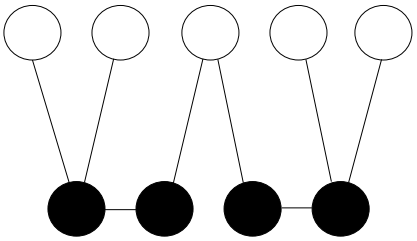


Figura B.2: Grafo não bipartite.

Códigos de bloco $\mathcal{C}(n,k)$ podem ser facilmente representados a partir de grafos. Foi Tanner [34] quem primeiro definiu um grafo bipartite - hoje chamado grafo de Tanner - relacionando os símbolos de uma palavra código com as equações de paridade que estes símbolos devem satisfazer de modo a serem uma palavra código válida.

Definição B.3 *(Grafo de Tanner)Um grafo de Tanner de um código de bloco C(n,k) é um grafo bipartite não-direcionado obtido a partir de um matriz de verificação de paridade **H** com N colunas e M linhas. O grafo de Tanner possui N **nós de símbolo** correspondentes às colunas de **H**, e M **nós de verificação** correspondentes às linhas de **H**. Um ramo conecta o nó de símbolo x_n ao m-ésimo nó de verificação c_m se e somente se $h_{m,n} = 1$, onde $h_{m,n}$ denota o elemento na m-ésima linha e na n-ésima coluna de **H**.*

Assim, pode-se construir um grafo de Tanner a partir da matriz de paridade **H** de um código linear. Como a matriz **H** de um código de bloco linear não é univocamente determinada, podemos ter vários grafos de Tanner distintos representando um mesmo código. O exemplo a seguir mostra a construção de um grafo de Tanner a partir da matriz de paridade de um código de bloco.

Exemplo B.1 *Considere o código de Hamming $\mathcal{C}(7,4)$ descrito pela seguinte matriz de verificação de paridade* **H***:*

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Uma palavra pertencente a este código $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ *deve satisfazer a seguinte condição:*

$$\mathbf{x}H^T = \mathbf{0}.$$

Então, as equações de paridade são:

$$x_1 \oplus x_2 \oplus x_4 \oplus x_5 = 0$$

$$x_1 \oplus x_3 \oplus x_4 \oplus x_6 = 0$$

$$x_2 \oplus x_3 \oplus x_4 \oplus x_7 = 0.$$

O grafo de Tanner para este código possui 3 *nós de verificação (representando as equações de paridade) e* 7 *nós de símbolo (representando os símbolos da palavra código). Denotando os nós de símbolo por* $\bigcirc$ *e o nós de verificação por* $\boxplus$*, o grafo de Tanner para* $\mathcal{C}(7,4)$ *é mostrado na Figura B.3.*

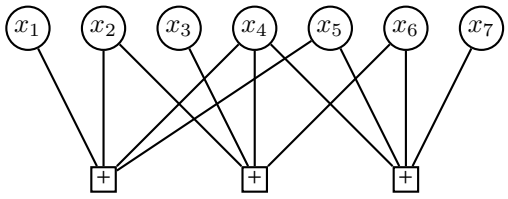
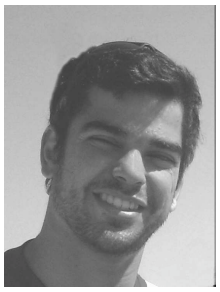


Figura B.3: Grafo de Tanner para o código de Hamming $\mathcal{C}(7,4)$.

Denotar-se-á o conjunto de nós de símbolo de um grafo de Tanner $\mathcal{G}$=(V,E) por $V_s = \{s_1, s_2, ..., s_N\}$ e o conjunto de nós de verificação por $V_c = \{t_1, t_2, ..., t_M\}$. Um conceito importante que foi utilizado diz respeito ao grau de um símbolo codificado. Em um grafo de Tanner, o grau de um nó de símbolo (ou de verificação), corresponde ao número de ramos que incidem neste nó e é denotado por $d(s_n)$ (ou $d(t_m)$).

Um outro conceito importante em teoria de grafos é o conceito de árvores. Uma árvore nada mais é do que um grafo conectado que não possui ciclos. A definição de árvores e ciclos tem grande importância na análise da probabilidade assintótica de falha do processo de decodificação de códigos LT.

SOBRE O AUTOR



O autor nasceu em Recife, Pernambuco, no dia 19 de outubro de 1981. Formado em Engenharia Elétrica, modalidade Eletrônica, pela Universidade Federal de Pernambuco em 2005, desde 2006 é membro da Sociedade Brasileira de Telecomunicações. Seus interesses de pesquisa incluem códigos para controle de erros, comunicações móveis, redes de computadores e teoria da informação.

Endereço: Rua Francisco Figueiroa, 184
Bairro Novo
Olinda – PE, Brasil
C.E.P.: 53.030 – 290

e-mail: hvbneto@hotmail.com

Esta dissertação foi diagramada usando LaTeX 2_ε[1] pelo autor.

[1] LaTeX 2_ε é uma extensão do LaTeX. LaTeX é uma coleção de macros criadas por Leslie Lamport para o sistema TeX, que foi desenvolvido por Donald E. Knuth. TeX é uma marca registrada da Sociedade Americana de Matemática ($\mathcal{AMS}$). O estilo usado na formatação desta dissertação foi escrito por Dinesh Das, Universidade do Texas. Modificado em 2001 por Renato José de Sobral Cintra, Universidade Federal de Pernambuco, e em 2005 por André Leite Wanderley.