UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Avaliação de Técnicas de Compressão de Sinais Para o Fronthaul

Flávio Mendes de Brito

DM: 43/2019

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2019

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Flávio Mendes de Brito

Avaliação de Técnicas de Compressão de Sinais Para o Fronthaul

DM: 43/2019

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2019

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Avaliação de Técnicas de Compressão de Sinais Para o Fronthaul

Dissertação apresentada ao comitê de pós-graduação do departamento de Engenharia Elétrica da Universidade Federal do Pará como requerimento para obtenção do grau de Mestre em Engenharia Elétrica com ênfase em Telecomunicações.

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2019

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

M538a Mendes de Brito, Flávio
Avaliação de Técnicas de Compressão de Sinais Para o Fronthaul / Flávio Mendes de Brito. — 2019.
70 f. : il. color.

Orientador(a): Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior
Dissertação (Mestrado) - Programa de Pós-Graduação em Engenharia Elétrica, Instituto de Tecnologia, Universidade Federal do Pará, Belém, 2019.

1. Compressão. 2. Fronthaul. 3. TCQ. 4. Quantização. 5. Cloud-RAN. I. Título.

CDD 621.3822

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

"AVALIAÇÃO DE TÉCNICAS DE COMPRESSÃO DE SINAIS PARA O FRONTHAUL"

AUTOR: FLAVIO MENDES DE BRITO

DISSERTAÇÃO DE MESTRADO SUBMETIDA À BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, SENDO JULGADA ADEQUADA PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE TELECOMUNICAÇÕES.

APROVADA EM: 27/11/2019

BANCA EXAMINADORA:

Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior
(Orientador – PPGEE/UFPA)

Prof. Dr. João Crisóstomo Weyl Albuquerque Costa
(Avaliador Interno – PPGEE/UFPA)

Prof. Dr. Leonardo Lira Ramalho
(Avaliador Externo ao Programa – FCT/UFPA)

VISTO:

Prof.ª Dr.ª Maria Emília de Lima Tostes
(Coordenadora do PPGEE/ITEC/UFPA)

Dedico este trabalho aos meu pai Miguel Salustiano de
Brito e a minha mãe Adelaide Mendes de Brito.

Agradecimentos

Gostaria de agradecer primeiramente a Deus por minhas realizações. Em segundo lugar, meus pai Miguel Salustiano de Brito e a minha mãe Adelaide Mendes de Brito e aos meus irmãos Franco Mendes de Brito e Fábio Mendes de Brito por todo o apoio que foi fundamental para esta conquista.

Em terceiro lugar, ao meu orientador professor Aldebaro Klautau por ter me aceitado como seu aluno de mestrado e por participar da banca me dado orientações que certamente levarei para o resto da minha vida.

Em quarto lugar, gostaria de agrader aos membros da banca: o Prof. Dr. João Crisóstomo Weyl Albuquerque Costa e o Prof. Dr. Leonardo Lira Ramalho pelas valiosas contribuições que certamente ajudarão a aprimorar este trabalho.

Em quinto lugar, gostaria de agradecer a todos os integrantes do Núcleo de Pesquisa e Desenvolvimento em Telecomunicações, Automação e Eletrônica (LASSE) pelo ótimo ambiente de trabalho e por todos os amigos que tive o privilégio de conhecer. Entre esses amigos, gostaria de agradecer especialmente a Ingrid Nascimento e ao Cleverson Nahum por terem ouvidos todas a minhas lamúrias que, sem dúvidas, dariam material suficiente para um livro apócrifo intitulado "Lamentações do Flávio".

Gostaria de agradecer também ao meu amigo e professor de piano Sérgio Sena por me mostrar o mundo maravilhoso que é o mundo da música e por acreditar que eu posso fazer parte desse mundo.

Finalizando, gostaria de agradecer a todas as pessoas que influenciaram direta ou indiretamente na conclusão deste trabalho. Muito obrigado a todos!

Flávio Mendes de Brito
Novembro 2019

Alegria, mais belo fulgor divino.
Filha de Elíseo,
Ébrios de fogo entramos
Em teu santuário celeste!
Tua magia volta a unir
O que o costume rigorosamente dividiu.
Todos os homens se irmanam
Onde pairar teu voo suave.

Friedrich Schiller

Lista de Figuras

Lista de Tabelas

Sumário

Abstract

The growing data demand of mobile networks has motivated the creation and evolution of architectures aiming to supply such transfer requirements. To meet these requirements, a number of challenges need to be met, including data transfer at the link between the Base Station Unit (BBU) and the Remote Radio Head (RRH). Known as fronthaul, this link requires high speed information transfer and one method that allows to tranfer more data using the same rate is data compression. Therefore, this study aimed to evaluate different techniques used in fronthaul data compression. Initially, the efficiency of some quantizers such as the scalar quantizer (SQ), two-dimensional vector (VQ) and the Trellis Coded Quantization (TCQ) was verified. The analysis consists of combining these quantizers with resampling, Block Scaling and Huffman coding. In both analyzes, it was found that the system using TCQ as quantizer obtained the best relationship between Error Vector Magnitude (EVM) and computational cost, offering an EVM lower than the scalar quantizer and a computational cost lower than the vector quantizer.

Index terms— Compression, Fronthaul, TCQ, Quantization, Cloud-RAN

Resumo

A crescente demanda de dados das redes móveis motivou a criação e evolução de arquiteturas objetivando suprir tais requisitos de transferência. Para suprir estas demandas, diversos desafios precisam ser vencidos e, entre eles, destaca-se a transferência de dados no link entre a Base Station Unit (BBU) e a Remote Radio Head (RRH). Conhecido como fronthaul, este link necessita de alta velocidade na transferência de informações e um método que permite transmitir mais dados usando a mesma taxa é chamado de compressão. Portanto, este trabalho teve como objetivo a avaliação de diferentes técnicas utilizadas na compressão de dados no fronthaul. Inicialmente, verificou-se a eficiência de alguns quantizadores tais como o quantizador escalar (SQ), vetorial de duas dimensões (VQ) e a Trellis Coded Quantization (TCQ). A análise consistiu na combinação destes quantizadores com resampling, Block Scaling e codificação Huffman. Nestas análises, verificou-se que o sistema utilizando TCQ como quantizador obteve a melhor relação entre Error Vector Magnitude (EVM) e custo computacional, oferecendo uma EVM inferior ao quantizador escalar e um custo computacional inferior ao quantizador vetorial.

Palavras-chave — Compressão, Fronthaul, TCQ, Quantização, Cloud-RAN

Capítulo 1

INTRODUÇÃO

O ser humano sempre desenvolveu métodos para melhorar a comunicação. Ao longo dos anos, diversas tecnologias foram desenvolvidas com o intuito de possibilitar a comunicação entre duas ou mais pessoas separadas fisicamente e a telefonia celular representou um avanço neste sentido. Através do desenvolvimento de arquiteturas que foram continuamente evoluindo até atingir os modelos que são utilizados atualmente (podendo citar as redes 4G e 5G), diversos problemas surgiram e foram solucionados através do uso criativo da engenharia. Entretanto, como o avanço das redes móveis é um processo contínuo, novas metas foram estipuladas e novos problemas surgiram. Um destes problemas diz respeito ao tráfego no link de comunicação entre a BBU e RRU atualmente conhecido como *fronthaul*.

Neste capítulo introdutório será apresentado um breve histórico das redes móveis e o que levou ao desenvolvimento das arquiteturas LTE e das redes C-RAN. Também será descrito a constituição de um sinal LTE no nível da camada física bem como o problema de tráfego no *fronthaul* que motivou a criação deste trabalho.

1.1 História das redes móveis

A primeira geração (1G) teve início em meados da década de 80. Nesta geração, o sistema era principalmente analógico. Este sistema atingia taxas baixíssimas se comparadas aos dias atuais. Pelo fato dos dispositivos móveis serem bastante caros na época, o uso desta tecnologia era bastante restrito [1], [2].

Somente com o advento da segunda geração das redes móveis (2G), o consumidor final pode ter largo acesso às redes móveis. Uma das principais novidades desta geração é o uso do processamento digital de sinais. Isto permitiu um aproveitamento melhor do espectro e também possibilitou a fabricação de dispositivos menores. Outra característica importante das redes 2G é que elas foram originalmente designadas para o uso exclusivo de voz. Entretanto, com o contínuo aprimoramento, foi possível introduzir a transmissão das messagens de texto ou *Short Message Service* (SMS). O sistema de rede 2G mais popular é conhecido como *Global System for Mobile Communications* (GSM).

Com o advento da internet ocorrendo na mesma época, as operadoras decidiram incorporar os serviços fornecidos pela rede mundial de computadores, possibilitando ao usuário a opção de realizar *download* de dados nos seus próprios dispositivos móveis. Esta evolução ficou conhecida como 2.5G e, para acompanhar o contínuo aprimoramento da taxa de transmissão de

dados da internet, as operadoras incorporam também a tecnologia conhecida como *Enhanced Data Rates for GSM Evolution (EDGE)*.

A terceira geração (3G) das redes móveis continuou o aprimoramento das redes celulares ao introduzir uma nova técnica de transmissão do sinal de rádio, permitindo um maior aprimoramento do uso do espectro. Esta tecnologia ficou conhecida como *Universal Mobile Telecommunication System* (UMTS). Uma vantagem do uso do UMTS foi a desnessidade de alterar a arquitetura física das antigas redes 2G. Entretanto, as redes 3G se popularizaram somente a partir de 2005 com a introdução das tecnologias *high speed downlink packet acess* (HSDPA) e *uplink packet access* (HSUPA). O uso coletivo da HSDPA e HSUPA é conhecido como *high speed packet access* (HSPA).

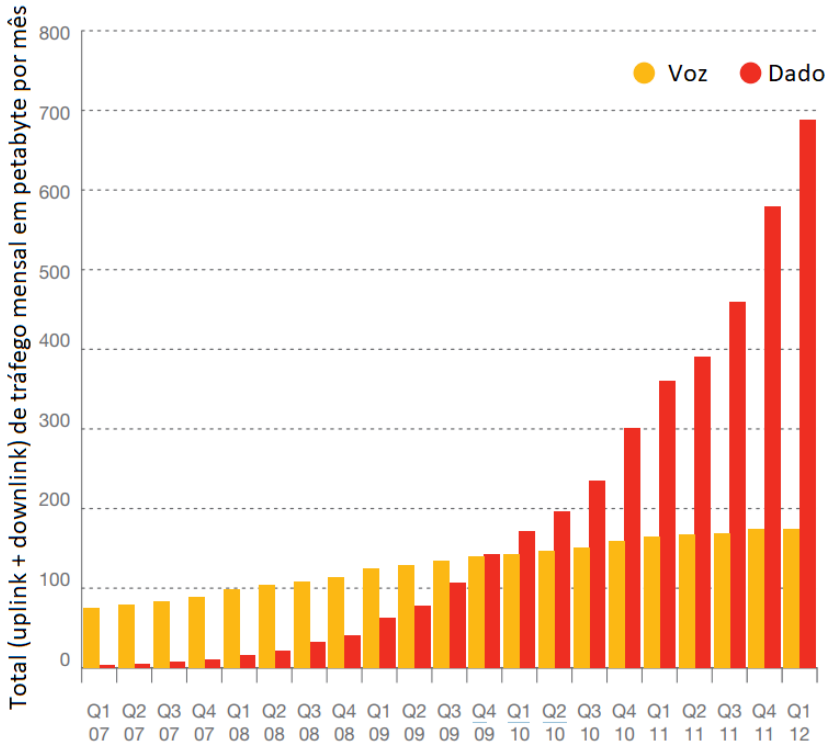
Em relação à tecnologia UMTS, ela pode ser implementada de duas maneiras: *Wideband code division multiple access* (WCDMA) e *Time division synchronous code division multiple access* (TD-SDCMA). O WCDMA foi o sistema originalmente desenvolvido para as redes 3G e é o mais difundido ao redor do mundo. O TD-SDCMA foi desenvolvido pela operadora chinesa *China Mobile* e foi uma maneira do governo chinês não depender da tecnologia do Ocidente bem como realizar pagamentos de *royalties*.

Existem, ao todo, duas principais diferenças entre o WCDMA e o TD-SDCMA. A primeira consiste no fato de que o WCDMA utiliza uma largura de banda de 5 MHz enquanto o TD-SDCMA utiliza 1.6 MHz. A segunda diferença é que o WCDMA separa as estações rádio base e os sistemas de transmissão utilizando *Frequency Division Duplex* (FDD) enquanto o *TD-SDCMA* utiliza *Time Division Duplex* (TDD). Ambas permitem que múltiplos usuários compartilhem o sistema sendo que FDD separa os usuários através do espectro enquanto o TDD separa os usuários através do tempo.

É importante notar que o tráfego de voz era predominante nesta época. Entretanto, a partir de 2010, o tráfego de dados começou a crescer substancialmente. A Figura 1.1 apresenta um gráfico que mostra uma comparação entre o consumo de dados e voz entre os anos de 2007 e 2011. Nota-se o fato de que o tráfego de dados aumentou consideravelmente em relação ao de voz [3].

Este crescimento se deu por fatores como o uso do 3.5G e a popularização de dispositivos como o *iPhone* em 2007 e o sistema operacional *Android* em 2008, por possuírem uma interface mais *user-friedly* e o suporte a criação de aplicativos desenvolvidos por terceiros, permitiu também um aumento e popularização ainda maior desses dispositivos. Com a adição do incentivo dados pelas operadoras (como o consumo ilimitado de dados de *downlinks*), a taxa de transmissão de dados continuava a crescer. Como ilustração da preocupação com o aumento contínuo do tráfego de dados, a Figura 1.2 mostrou a projeção realizada pela Ericson em 2011 que mostrou como seria o tráfego de dados e voz pelo mundo entre os anos de 2011 e 2016 e a Figura 1.1 mostra o real tráfego obtido nos anos de 2013 a 2019. Nota-se que as previsões realizadas pela Ericson (Figura 1.1) se mostraram confiáveis pois o tráfego continuou a crescer e a diferença entre o tráfego de dados e o de voz continou a aumentar como é visto na Figura 1.3. É interessante notar no gráfico da Figura 1.3 a linha preta que indica a taxa de crescimento ao ano. Para analisar o gráfico, deve-se comparar cada respectivo quadrimestre. Por exemplo, entre o segundo quadrimestre de 2018 e o segundo quadrimestre de 2019, houve um crescimento aproximado de 78% do tráfego de dados e voz e outro fator interessante a se notar é o crescimento exponencial do tráfego de dados (caracteristica que tende a aumentar com a difusão das redes 5G ao redor do mundo [4]).

Figura 1.1: Tráfego de dados e voz no mundo no período correspondente de Janeiro de 2007 a Janeiro de 2012.



Fonte: adaptado de [1].

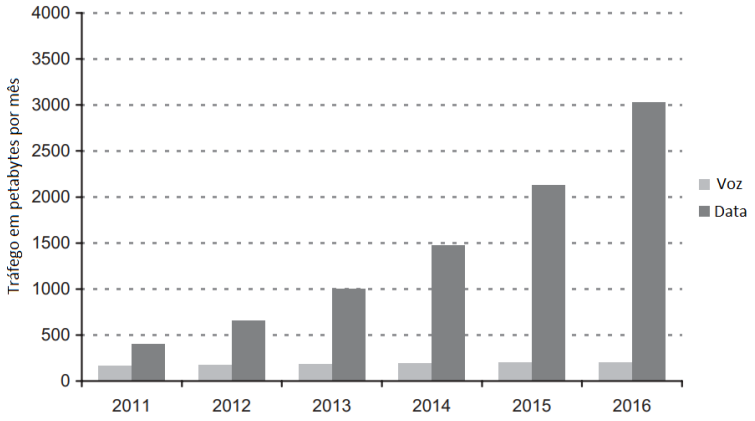
1.2 Arquitetura LTE

Todas essas demandas de taxa de dados levaram ao desenvolvimento de uma arquitetura que fosse competitiva pelos próximos 10 anos no mínimo. O estudo desta evolução teve como ponto principal suprir os requerimentos da interface de ar. Esta evolução levou ao desenvolvimento da arquitetura LTE e nesta seção será apresentado os principais pontos deste modelo.

Os principais componentes da arquitetura LTE são o *User Equipement* (UE), *Evolved UMTS terrestrial radio access network (E-ULTRAN) [7] e [8]* e o *Evolved Packet Core* (EPC) [9]. O UE consiste em um dispositivo móvel, como um celular por exemplo, que se comunica com o E-ULTRAN que, por sua vez, envia a informação para o EPC que se comunica com o mundo externo. A comunicação entre um dispositivo e outro desta arquitetura é realizada através de protocolos. O protocolo utilizado entre o UE e o E-ULTRAN é conhecido como Uu; o protocolo utilizado para a comunicação entre a E-ULTRAN e o EPC é conhecido como S1 e, finalizando, o protocolo utilizado entre o EPC e o tráfego externo é conhecido como SGi. A Figura 1.4 apresenta os três blocos listados (UE, E-ULTRAN e EPC) bem como o protocolo utilizado por cada um deles (Uu, S1 e SGi).

A seguir, serão apresentados em maiores detalhes os três componentes principais das redes LTE discutidas anteriormente.

Figura 1.2: Projeção de tráfego de dados e voz no mundo no período de 2011 a 2016.



Fonte: adaptado de [5].

1.2.1 *User Equipment* (UE)

Basicamente, o UE é o dispositivo que o usuário final utiliza (pode ser um celular, por exemplo). O UE consiste de dois componentes principais: o *Mobile Equipement* (ME) e o *Universal Integrated Circuit Card* (UICC). O ME, por sua vez, se divide em dois componentes: o *Terminal Equipment* (TE) e *Mobile Termination* (MT). O TE é responsável pela transmissão dos dados dentro da UE enquanto o MT é responsável por lidar com todas as funções relacionadas a comunicação.

O UICC, mais conhecido por cartão SIM, é responsável por armazenar informações de identificação do usuário (como o número do telefone, por exemplo). Estas informações são armazenadas através de uma aplicação conhecida como *Universal Subscriber Identity Module* (USIM) [10]. É importante destacar o LTE tem suporte somente para versão *Release 99* or superior desta aplicação.

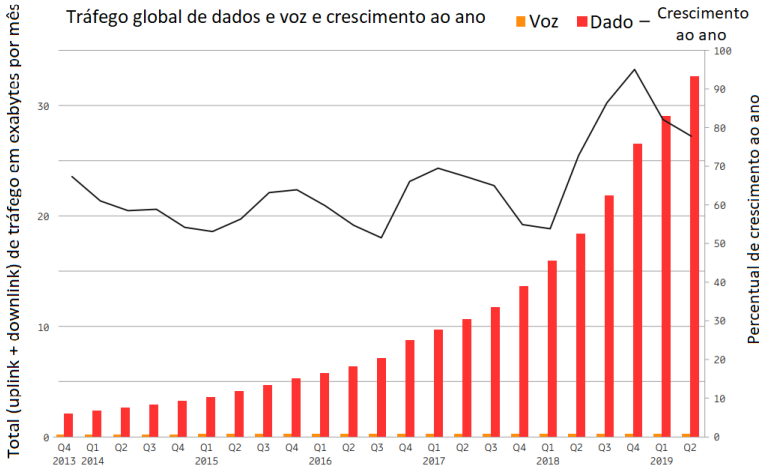
Para ilustrar, a Figura 1.5 apresenta um diagrama do UE. Nesta Figura, nota-se novamente que o UE consiste de dois componentes: ME e UICC. o ME, por sua vez, consiste em mais dois componentes: TE e MT [11].

1.2.2 *Evolved UMTS Terrestrial Radio Access Network* (E-UTRAN)

O E-UTRAN é responsável por gerenciar todo o tráfego de rádio entre o dispositivo móvel e o *Evolved Packet Core* (EPC) [12]. Ao contrário do UE, o E-UTRAN só possui um componente que é conhecido como *Evolved Node B* (eNB). Uma eNB possui basicamente duas funções. A primeira consiste em receber os sinais de *downlink* e enviar os sinais de *uplink* utilizando todo o sistema de processamento de sinais disponíveis da interface de ar do sistema LTE. A segunda funçao de uma eNB consiste em gerenciar as operações de cada dispositivo móvel em um nível de abstração baixo através do envio de mensagens que informam as caracteristicas da interface de ar utilizada, por exemplo. Cada eNB consiste em uma *base estation*.

Além da eNB, existe o que é conhecido como *home eNB* (HeNB) que é uma estação

Figura 1.3: Tráfego de dados e voz no período de 2013 a 2019.



Fonte: adaptado de [6].

rádio base que pode ser comprada pelo usuário para o uso de *fentocells* dentro de uma área bem restrita (como sua própria casa, por exemplo).

1.2.3 *Evolved Packet Core* (EPC)

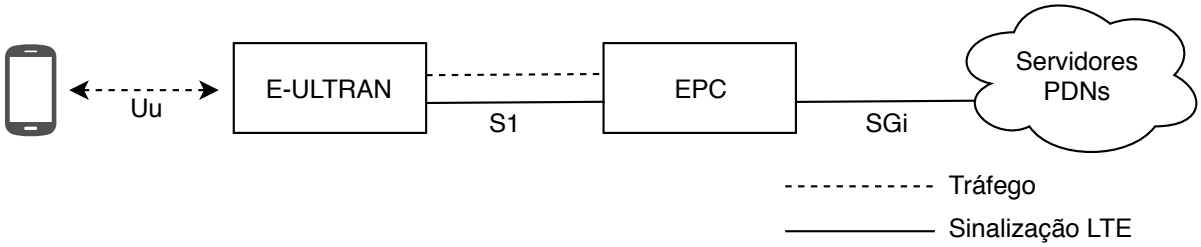
Como explicado anteriormente, o EPC é responsável por realizar a comunicação entre o sistema LTE e o meio externo. A Figura 1.6 apresenta uma descrição gráfica do EPC. Nota-se que ele possui vários componentes importantes. O P-GW é o componente responsável por gerenciar o tráfego entre o EPC e o mundo externo através da interface SGi. O SG-W (*serving gateway*), por sua vez, atua na função de roteamento entre a eNB e o PDN.

O *Mobility Management Entity* (MME) é responsável por gerenciar as operações dos dispositivos móveis em um alto nível de abstração (abstração acima da camada de rádio). O MME realiza este gerenciamento através do envio de mensagens que tratam de questões importantes tais como segurança e o gerenciamento do tráfego de dados que não estão relacionados a transmissão de rádio. Cada dispositivo móvel pode se conectar a um MME, o que é chamado de *serving MME*. Entretanto, esta conexão pode ser alterada caso o usuário se afaste da eNB. Finalizando, outra função importante do MME consiste em gerenciar outros elementos que são internos ao EPC.

Outro componente do EPC é conhecido como *Home Subscriber Server* (HSS) que é um banco de dados que contém informações sobre os usuários da rede. O HSS é um dos componentes que foram aproveitados do UMTS e do GSM.

Finalizando, o EPC também contém outros componentes não apresentados na Figura 1.6. Entre estes componentes, pode-se citar o *Cell Broadcast Center* (CBC) e o *Cell Broadcast Service* (CBS).

Figura 1.4: Diagrama geral de uma arquitetura LTE.



Fonte: apdatado de [1].

1.3 Gerações das redes móveis

No início do desenvolvimento dos sistemas móveis, todo o processamento necessário até a geração do sinal a ser transmitido pelo ar era realizado em uma única unidade de Processamento. A Figura 1.7 apresenta uma descrição do modelo utilizado em que o sistema realiza todo o processamento em banda base e em Rádio Freqüência (RF) [13] em um único local e o sinal é transmitido até a antena através de um cabo coaxial.

Um problema gerado por esta arquitetura é a perda de sinal entre a Unidade de processamento de sinal e a antena de transmissão. Um avanço para combater esta perda consistiu em uma mudança na arquitetura em que o sinal a ser transmitido se localiza mais próximo da antena de transmissão. Neste caso, houve o acréscimo do uso de fibra ótica para transmitir o sinal em banda base até a Unidade de processamento de Rádio Freqüência. O sinal RF é, por sua vez, transmitido até a antena através do cabo coaxial (cujo comprimento foi reduzido). A Figura 1.8 apresenta a descrição deste segundo cenário apresentado.

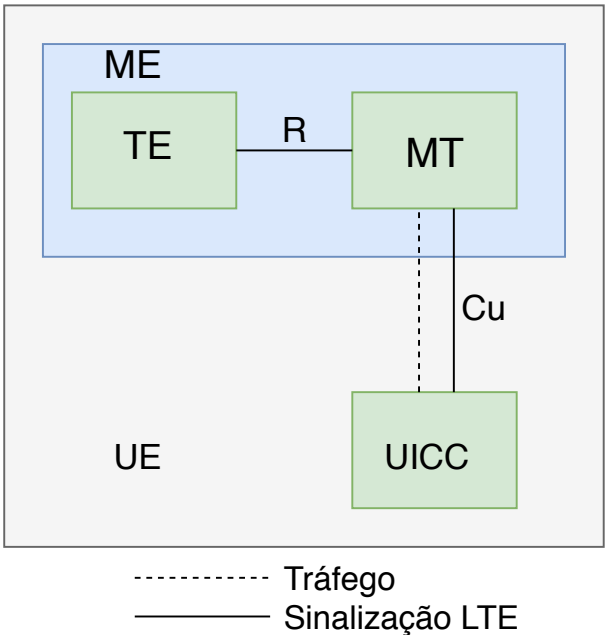
Com o passar do tempo, a indústria notou que unir diversas BBUs em um único local iria permitir reduzir os custos de manutenção. Como exemplo, reunir todas essas unidades de processamento permite que um único sistema de refrigeração opere sobre as BBUs reunidas. Há também o compartilhamento de fornecimento de energia.

Esta arquitetura é conhecida como RAN centralizada e permitiu uma grande economia de energia. Como ilustrado em [14], foi possível observar uma redução de 41 de consumo de energia apenas através do compartilhamento de ar-condicionado. Neste trabalho, 21 estações rádio base foram agrupadas para fornecer o processamento necessário para transmitir os sinais através de 101 antenas.

Um exemplo de arquitetura de RAN Centralizada é mostrada na Figura 1.9 em que as BBUs são organizadas em um único local e cada antena é conectada diretamente a unidade central contendo todas as BBUs. Nesta arquitetura, as BBUs são conectadas através de links óticos de alta velocidade e o link que conecta a BBU com a RRU é chamado *fronthaul*. A Figura 1.9 apresenta uma descrição da arquitetura de RAN centralizada.

A próxima mudança na arquitetura das redes móveis levou ao que é conhecido como C-RAN (*Cloud Radio Acess Network*). Esta arquitetura é uma evolução da RAN centralizada no sentido de que agora BBU e sua respectiva RRU não estão conectadas diretamente como foi mostrado na Figura 1.9 mas sim o processamento para cada antena é compartilhado entre as BBUs. A Figura 1.10 apresenta uma representação da arquitetura C-RAN. Nota-se nesta figura

Figura 1.5: Arquitetura do *User Equipment*.



Fonte: apdatado de [1].

que as antenas não estão conectadas diretamente a sua respectiva BBU.

Uma das vantagens da arquitetura C-RAN consiste justamente no compartilhamento de processamento de BBUs o que permite uma otimização no consumo de energia em relação à arquitetura de RAN centralizada. A Figura 1.10 apresenta uma ilustração da arquitetura C-RAN. Nota-se que a RRU não conecta-se diretamente a uma BBU específica, mas essa conexão é abstraída justamente para que o poder computacional da BBUs seja compartilhado.

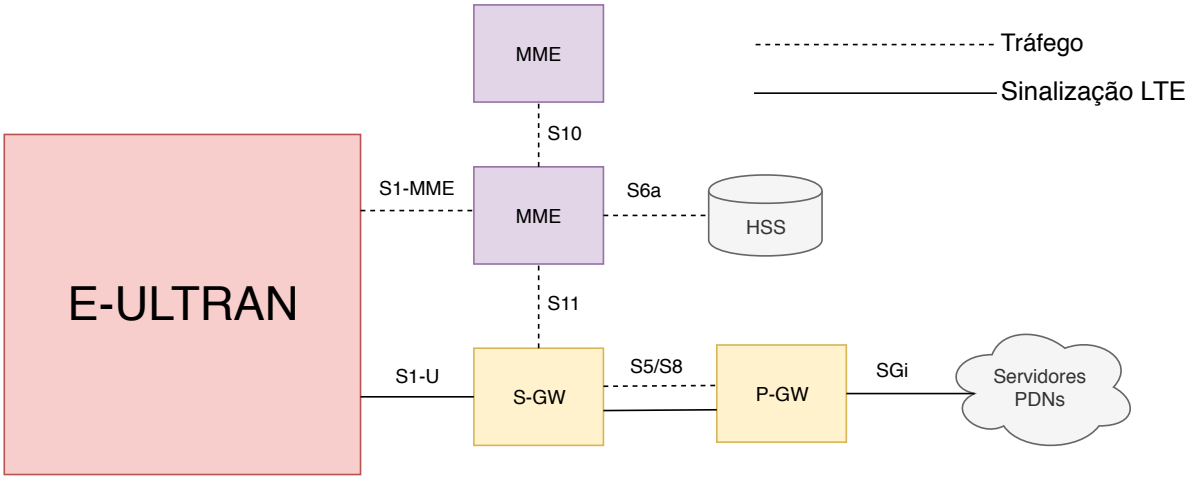
1.4 Descrição da interface de ar de um sinal LTE

Nesta seção, será apresentado uma descrição da constituição e funcionamento da interface de ar de um sinal LTE no nível da camada física. Serão apresentados os principais canais que definem o sinal LTE bem como a definição das principais configurações a serem utilizadas na interface de ar de um sinal LTE.

Esta seção inicia pela descrição da pilha de protocolos da interface de ar. Esta descrição é visualizada na Figura 1.11 que descreve os principais blocos utilizados nesta etapa. No topo da pilha, há dois blocos principais: o plano de usuário e plano de controle. No plano de usuário, a aplicação inicia o fluxo de dados que são transferidos através dos protocolos TCP e UDP e o protocolo IP. Entretanto, no plano de controle, a RRC (*Remote Radio Control*) é responsável por criar os sinais de controle que são trocados entre a estação rádio base e o dispositivo móvel [1].

Ambos os sinais criados pelo plano da aplicação e pelo plano de controle são processados pelo protocolo de convergência de plano de dados (PDCP); o link de controle de rádio (RLC) e o protocolo MAC até finalmente atingir a camada física da pilha de protocolos antes de ser

Figura 1.6: Arquitetura do *Evolved Packet Core* (EPC).



Fonte: apdatado de [1].

transmitida pelo ar que é constituída de três blocos de processamento [1] ilustrados a seguir:

- **bloco de processamento do canal de transporte:** responsável por aplicar códigos de correção de erros.
- **bloco de processamento do canal físico:** responsável pela modulação OFDMA, SC-FDMA e transmissão múltiplo de antenas ao sinal.
- **bloco de processamento analógico:** realiza a conversão do digital para analógico e realização de todo o processamento necessário para a transmissão do sinal pelo ar.

Além destes blocos, na Figura 1.11 há a presentação dos Canais Lógicos; de Transporte e de dados físicos. Descreve-se a seguir estes três canais listados em detalhes.

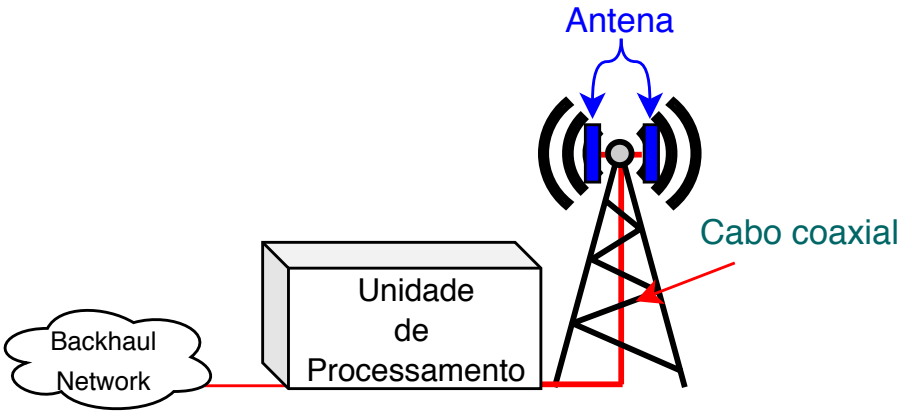
1.4.1 Canais lógicos

Estes canais se distinguem principalmente pelo tipo de dados que são transmitidos. Os canais de lógicos de tráfego transmitem os dados de usuários enquanto os canais lógicos de controle transmitem informações de controle. Outra característica destes canais é que os canais lógicos de tráfego não são compartilhados entre diferentes usuários enquanto os canais lógicos de controle podem ser compartilhados por mais de um usuário.

Os principais canais de transporte são o *dedicated traffic channel* (DTCH) e o *dedicated control channel* (DCCH). O primeiro é responsável por transmitir os dados do usuário enquanto o DCCH transmite as informações de controle.

Outros canais importantes dos canais lógicos é *paging control channel* (PCH) e o *broadcast control channel* (BCCH) que transmitem informações que devem ser transmitidas para todos os usuários da célula coberta.

Figura 1.7: Arquitetura das primeiras gerações das redes móveis.



Fonte: o autor.

1.4.2 Canais de transporte

O canal de transporte mais importante é o *uplink shared channel* (USCH) e o *downlink shared channel* (DSCH). Estes canais transportam a grande maioria dos sinais de dados e de controle (tanto do downlink com o DSCH quanto o uplink com USCH).

Outro canal importante é o *broadcast channel* (BCH) que transmite sinais que são provindos do *master information block*. O *paging channel* (PCH) transmitem as mensagem provenientes do *paging control channel* (PCCH).

1.4.3 Canais de dados físicos

Entre os canais de dados físicos mais importantes listam-se o *physical downlink shared channel* (PDSCH) e o *physical uplink shared channel* (PUSCH). O canal PDSCH transmite os sinais de downlink do usuário enquanto o PUSCH transmite os sinais de uplink.

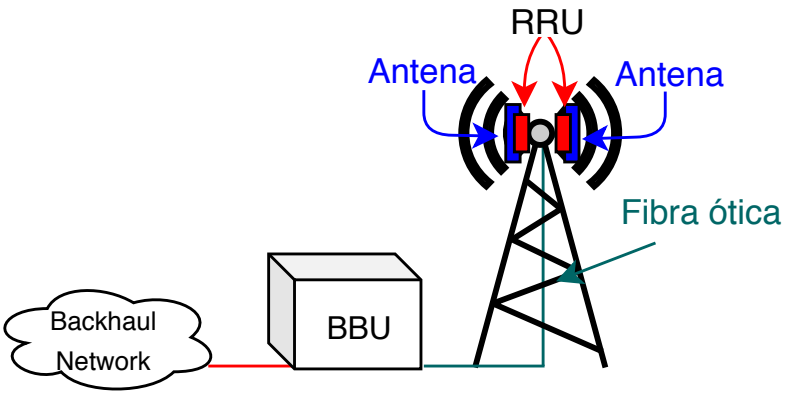
1.5 Prefixo cíclico, organização em slots e largura de banda

Estes canais listados anteriormente são mapeados através da modulação OFDMA. Através do prefixo cíclico, é adicionado uma redundância para diminuir o efeito da interferência intersimbólica (ISI) [15]. Há dois tipos de prefixo cíclico: o normal e o estendido.

Ao utilizar o prefixo cíclico, cada símbolo é precedido por um prefixo cíclico cuja duração pode ser de dois tipos: $4.7\mu s$. O prefixo cíclico estendido possui uma duração maior de $16.7\mu s$.

Para organizar todas estas informações, tanto o domínio do tempo quanto o da freqüência são utilizados. Esta organização é conhecida como *resource grid* que é composto por diferentes *resource blocks*. Cada *resource block*, por sua vez, é constituído de *resource elements* que é a menor unidade disponível nesta organização. Para melhor entendimento, a Figura 1.11 mostra a organização do *resource grid* do lado esquerdo e um exemplo de *resource block* é mostrado do

Figura 1.8: BBU e RRU são separadas.



Fonte: o autor.

lado direito. Nota-se na Figura 1.11 que cada *resource block* é composto de 7 símbolos OFDMS e cada símbolo OFDM é composto de 12 sub-portadoras.

Finalizando, para transmitir estes sinais, há algumas opções de largura de banda disponíveis: 20 MHz, 15 MHz, 10 MHz, 5 MHz, e 1.4 MHz.

1.6 Transmissão de sinal através do *Fronthaul*

Transmitir os sinais descritos anteriormente requer algoritmos de compressão e, especificamente neste trabalho, os sinais de *downlink* torna-se um desafio a medida que novas tecnologias, como os sistemas MIMO (*Multiple Input Multiple Output*) aumentam. Como exemplo, considere um sinal $x[n]$ com as seguintes características:

- **Largura de Banda (B):** 20 MHz.
- **Frequência de Amostragem (f_a):** 30.72 MHz.
- **Bits por componente real e imaginário (Q_s):** 15 bits.

A taxa total r em bits por segundo necessária para transportar o sinal $x[n]$ pelo *fronthaul* é:

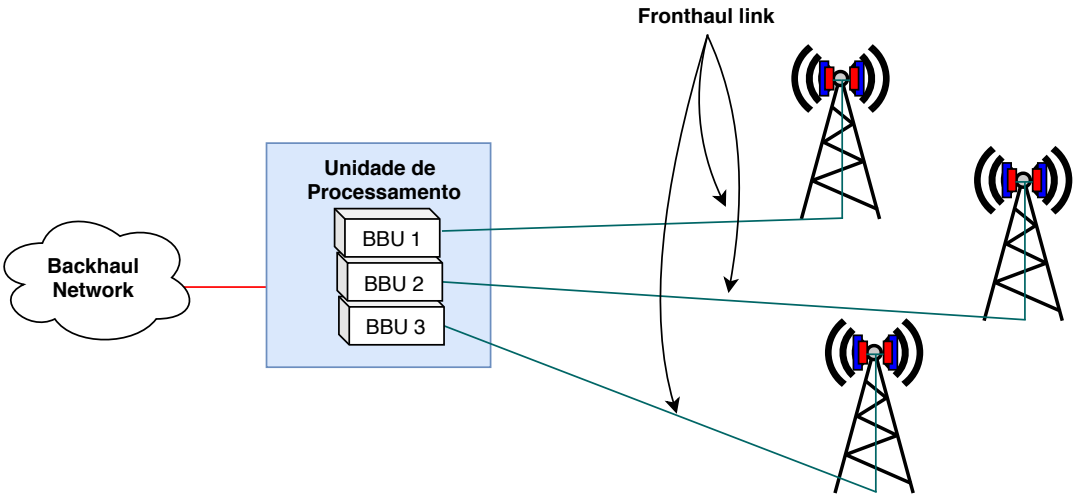
$$r = 2 \times Q_s \times f_a \tag{1.1}$$

Substituíndo os valores da Equação 1.1, temos:

$$r = 2 \times 15 \times 30.72 \times 10^6 \tag{1.2}$$

$$r = 921.6 Mbits/sec \tag{1.3}$$

Figura 1.9: Arquitetura de RAN centralizada.



Fonte: o autor.

Ou seja, para transportar apenas 1 frame de sinal LTE de 20 MHz é necessário uma taxa próxima de 1 Gigabits por segundo. Além disto, é importante frisar que este custo é para apenas uma antena. Com o advento das redes MIMO, esta taxa r aumenta consideravelmente.

Portanto, alternativas devem ser propostas para diminuir o tráfego de sinal pelo *fronthaul*. Uma destas alternativas é o uso da compressão de sinais que visa representar o sinal a ser transmitido com um menor número de recursos.

Este trabalho visa apresentar e avaliar diferentes técnicas de compressão de sinais. Os objetivos listam-se a seguir:

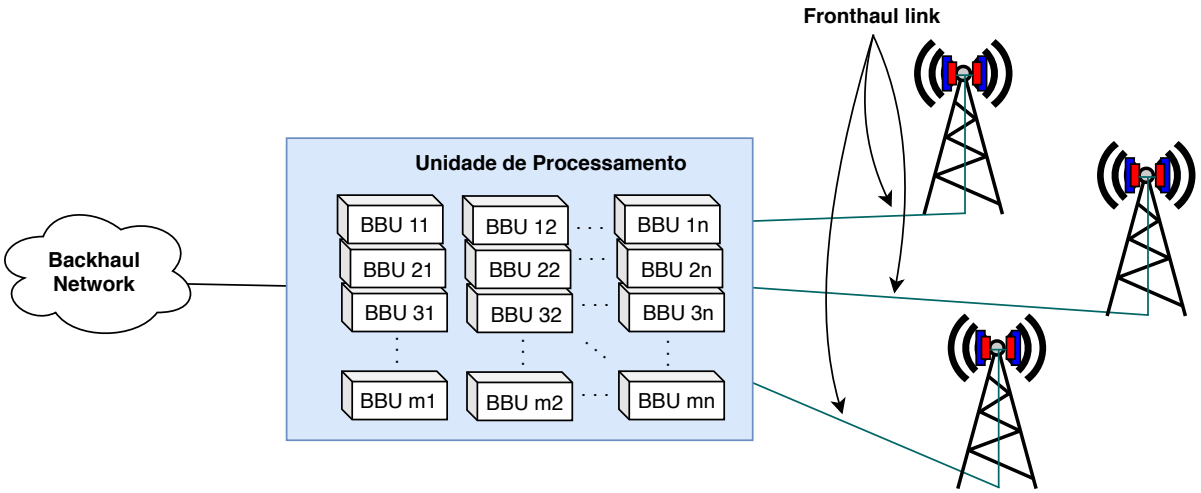
- Avaliar diferentes técnicas de compressão de fronthaul.
- Apresentar em detalhes a técnica de quantização conhecida como "*Trellis Coded Quantization*"e seu custo computacional aplicados ao *fronthaul*.
- Propor um novo sistema de compressão de sinais no *fronthaul* baseado no TCQ.

1.7 Organização do trabalho

Para um melhor entendimento do que está por vir neste documento, esta seção apresenta uma breve descrição de cada um dos capítulos seguintes. Este trabalho é organizado da seguinte forma:

- O capítulo **Fundamentos Teóricos** apresenta os conceitos de compressão e quantização que serão necessários neste trabalho.
- O capítulo **TCQ** apresenta o funcionamento do TCQ bem como seu custo computacional. Para melhor entendimento desta técnica, um exemplo numérico também é apresentado.

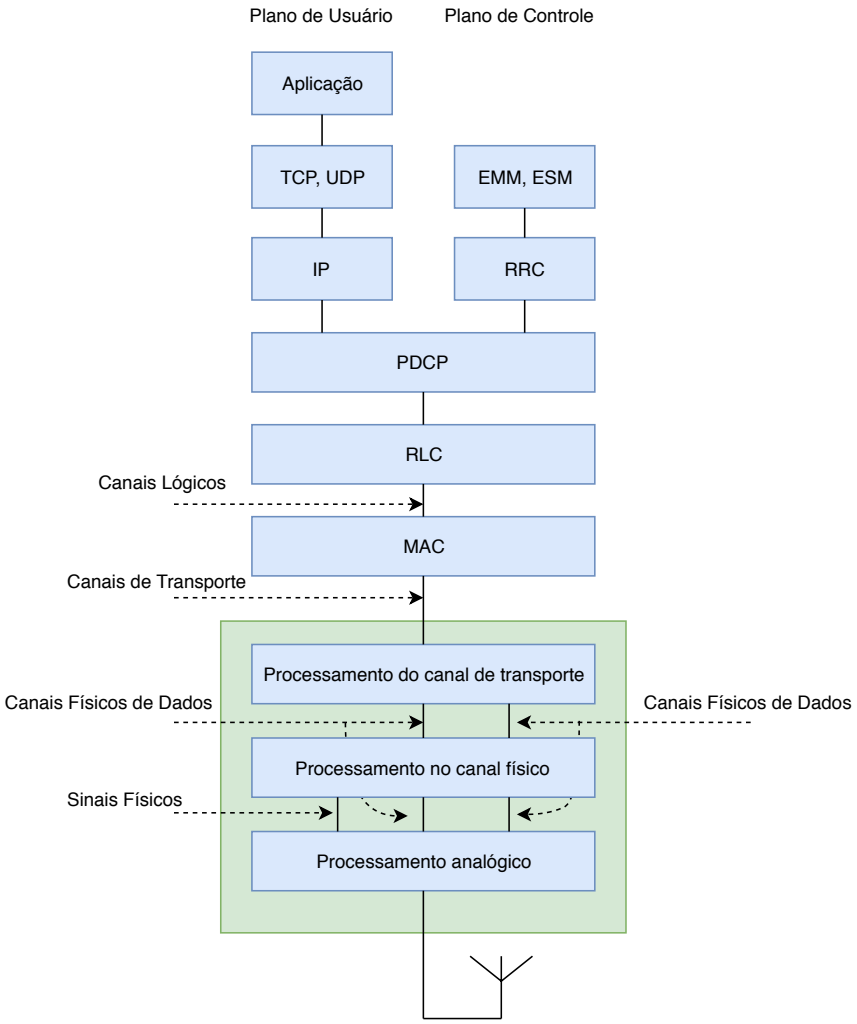
Figura 1.10: Arquitetura de RAN centralizada.



Fonte: o autor.

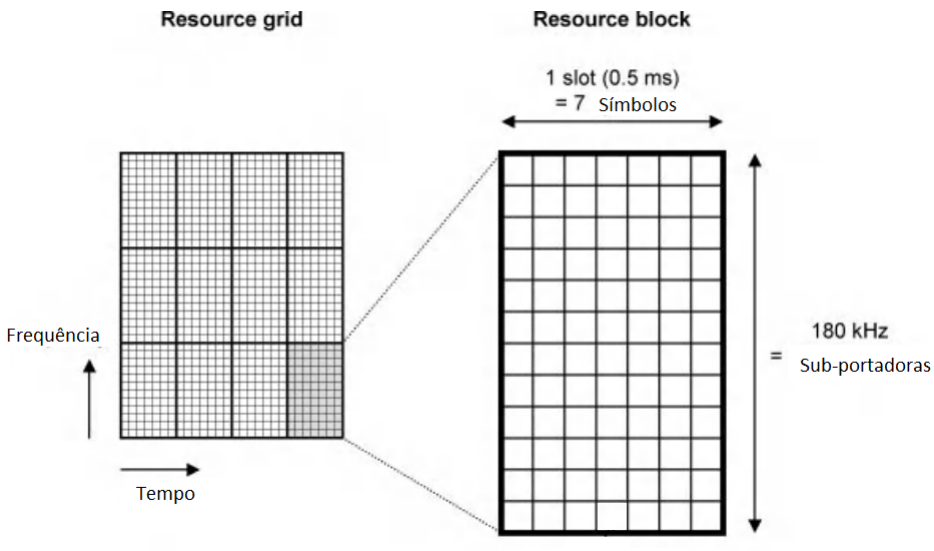
- O capítulo **Experimentos e Resultados** apresenta a descrição de cada um dos experimentos e mostra seus respectivos resultados.
- O capítulo **Conclusão e Trabalhos Futuros** conclui este texto e também apresenta trabalhos futuros a serem desenvolvidos.

Figura 1.11: Arquitetura da interface de ar.



Fonte: adaptado de [1].

Figura 1.12: Ilustração do *resource grid* e *resource block*.



Fonte: adaptado de [1].

Capítulo 2

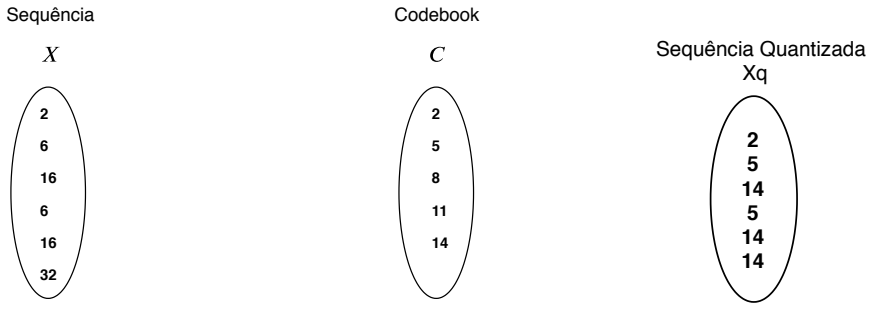
FUNDAMENTOS TEÓRICOS

Neste capítulo serão abordados os conceitos teórico necessários para projetar os algoritmos de quantização escalar, vetorial e o TCQ bem como apresentar os trabalhos relacionados utilizados para compressão de sinais do *fronthaul*.

2.1 Introdução a quantização

A quantização é uma técnica muito comum na área de processamento de sinais e consiste basicamente em selecionar, dentro de um conjunto chamado *codebook*, os elementos que representam uma sequência X com o menor erro possível. Como ilustração, a Figura 2.1 apresenta um exemplo de quantização escalar em que a sequência X é mostrada na Figura 2.2(a); o *codebook* C na Figura 2.2(b) e a seqüência quantizada Xq na Figura 2.2(c).

Figura 2.1: Ilustração de um quantizador genérico.



(a) Sequência a ser quantizada. (b) Codebook. (c) Sequência quantizada.

Fonte: o autor.

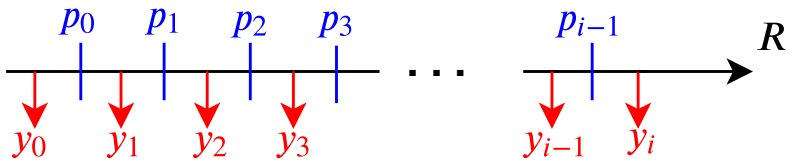
Um quantizador Q também pode ser definido como uma função que mapeia uma seqüên-

cia C em um codebook R, ou seja,

$$Q : C \rightarrow R. \tag{2.1}$$

Um quantizador também pode ser determinado pelos seus valores de saída y e pela sua **partição** p. Para exemplificar, a Figura 2.2 apresenta um quantizador genérico Q que divide a reta R em i partições e $i + 1$ valores de saída.

Figura 2.2: Quantizador genérico.



Fonte: o autor.

Nota-se na Figura 2.2 que os valores de saída do quantizador estão ordenados de forma crescente, ou seja,

$$y_0 < y_1 < y_2 < y_3 < y_{i-1} < y_i. \tag{2.2}$$

2.1.1 Quantizador regular e não regular

Um quantizador é classificado como regular se cada partição é um intervalo único e o valor de saída y_i está entre o intervalo p_{i-1} e p_i. Outra característica importante dos quantizadores regulares é que, se todos os valores a serem quantizadores estiverem entre o intervalo definido pelos pontos p_{i-1} e p_i, o valor de saída do quantizador será sempre y_i. A terceira característca dos quantizadores regulares é que os valores das partições estão ordenados crescentemente [16], ou seja,

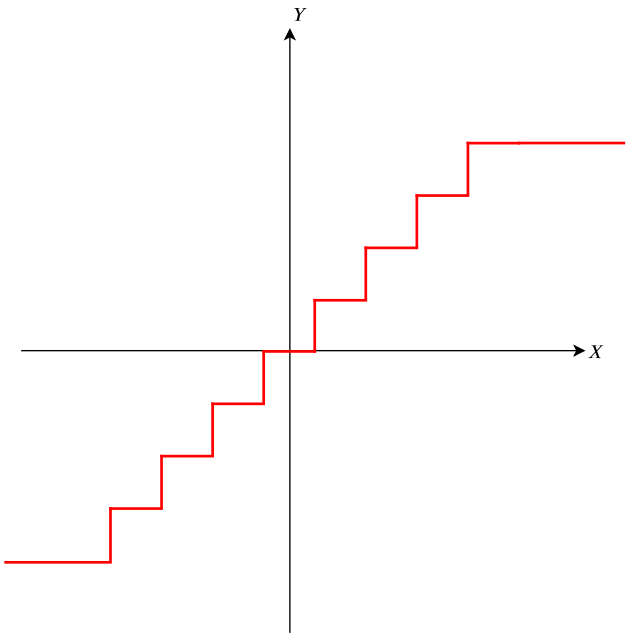
$$p_0 < p_1 < p_2 < ...p_{i-2} < p_{i-1}. \tag{2.3}$$

Uma forma de representar um quantizador regular é através do gráfico apresentado na Figura 2.6 abaixo em que cada valor da reta real representada pela abcissa X é mapeada em um valor específico da ordenada Y.

Os quantizadores também podem ser classificados em *mid-rise* e *mid-tread*. Quantizadores *mid-tread* são quantizadores regulares que possuem a presença do valor quantizado zero enquanto os quantizadores *mid-rise* não possuem o valor zero em seu codebook.

Há dois tipos de erros relacionados aos quantizadores que dependem da região em que o erro de quantização ocorre. A seguir, listam-se cada um desses erros.

Figura 2.3: Quantizador escalar.



Fonte: o autor.

2.2 Erro granular e erro *overload*

O erro granular consiste no erro que é encontrado quando um valor x a ser quantizado se encontra na região granular de um quantizador Q. Caso o valor a ser quantizado se encontre na região *overload* do quantizador, tem-se, portanto, o que é chamado de erro *overload*čitegersho2012vector . Para exemplificar, a Figura 2.4 apresenta as duas regiões de erro em que a região onde ocorre o erro granular é mostrado na Figura 2.5(a) e a região do quantizador que ocorre o erro *overload* é mostrado na Figura 2.5(b).

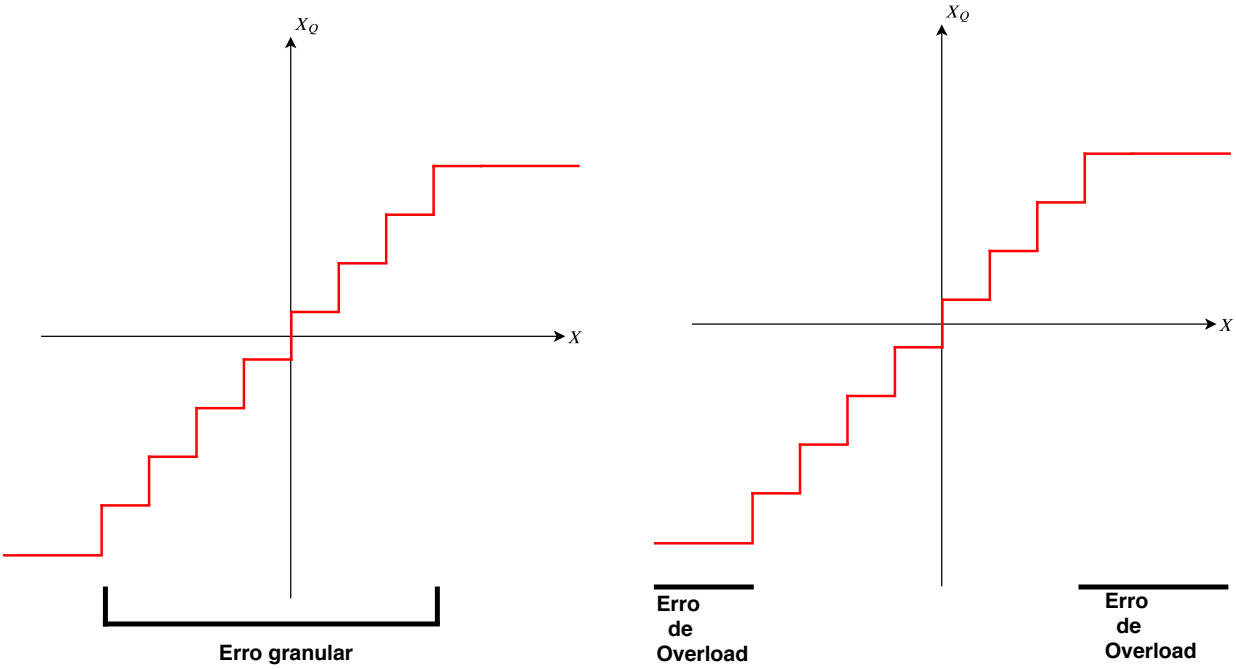
2.3 Otimizando os parâmetros de um quantizador

Um grande desafio no projeto de quantizadores diz respeito a definição do codebook e da partição a ser utilizados. Definir qual será o conjunto de partições composto pelos valores x_i e x_{i-1} e o correspondente valor de saída y_i para cada partição impacta diretamente na performance do sistema e no erro adquirido.

O primeiro passo para otimizar o projeto de um quantizador consiste em definir uma medida de erro que leve em conta a distribuição de probabilidade da fonte s. Como definido em [16], [17] e [18], uma medida de erro D pode ser definido como

$$D = \sum_{i=1}^{N} \int_{R_i} (x - y_i)^2 f_x(x) dx. \tag{2.4}$$

Figura 2.4: Error Granular e Overload.



(a) Erro Granular. (b) Erro Overload.

Fonte: o autor.

A Equação 2.4 acima pode ser interpretada com o auxílio da Figura 2.6. Basicamente, a reta dos reais x é dividida em N regiões. Calcula-se, então, a diferença quadrática entre o valor x e o valor quantizado y_i dentro da região N_i multiplicado pela probabilidade $f_x(x)$ e o valor de saída y_i.

Encontrar um conjunto de valores y_i e definir as regiões R_i que minimizam o erro D não é uma tarefa trivial. Entretanto, através de uma abordagem iterativa que será descrita logo a seguir, é possível projetar regiões R e pontos y_i que diminuem o erro D a um valor considerado aceitável na maioria dos casos. Para isto, uma boa opção é transformar o problema de minimização do erro D da Equação 2.4 nos seguintes problemas:

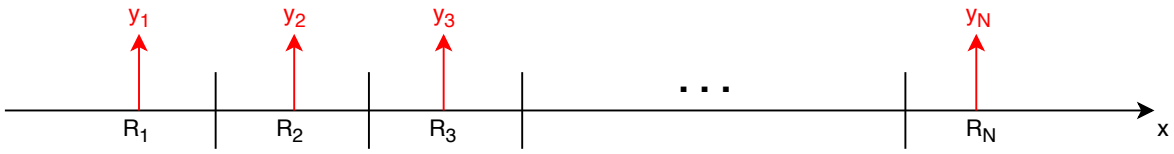
1. Encontrar a melhor partição para um codebook.
2. Encontrar o melhor codebook para uma partição.

Ambos os problemas apresentados acima serão explicados em detalhes a seguir.

2.3.1 Encontrar a melhor partição para um codebook

Este problema pode ser selecionado de maneira simples aplicando o princípio do vizinho mais próximo (*Nearest Neighbor*). Em outras palavras considere, como exemplo, o codebook apresentado na Figura 2.6. Como escolher o conjunto de pontos da partição x de tal maneira

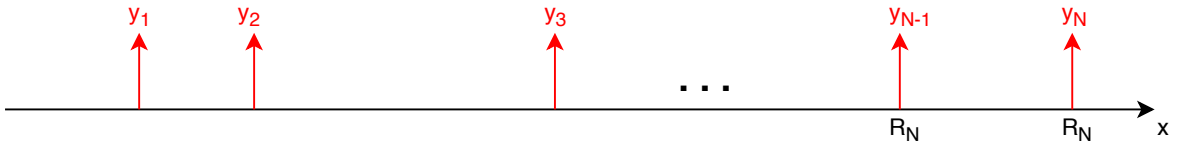
Figura 2.5: Reta dos reais particionada em N regiões.



Fonte: o autor.

a minimizar o erro de quantização? A resposta é justamente aplicando o princípio de *Nearest Neighbor*.

Figura 2.6: Como definir a partição para este codebook específico?



Fonte: o autor.

O princípio de *Nearest Neighbor* pode ser definido matematicamente da seguinte forma: seja um quantizador $Q(x)$ e a medida de erro entre duas variáveis x e y_i definida como $d(x, y_i)$. Então, as regiões R_i são definidas como

$$R_i \subset \{x : d(x, y_i) \leq d(x, y_j); j \neq i\}. \tag{2.5}$$

A Equação 2.5 acima nos diz que uma Região R_i será aquela que contém todos os valores x em que a distância entre o x e o ponto y_i é menor do que a distância entre o valor x e todos os outros valores y_j.

Em outras palavras, os codebooks definidos pelo quantizador $Q(x)$ serão definidos como

$$Q(x) = y_i \text{ somente se } d(x, y_i) \leq d(x, y_j) \text{ para todo } j \neq i. \tag{2.6}$$

A Equação acima informa que o valor x será quantizado com o valor y_i se, e somente se, o erro $d(x, y_i)$ for o menor entre x e y_i do que entre x e y_j (que são os valores restantes de y).

Para minimizar o erro utilizando o princípio do vizinho, aplica-se a seguinte regra: se o valor x a ser quantizado estiver entre duas *codewords* (y_{i-1} e y_i), o melhor valor da partição p_{i-1} que separa os valores y_{i-1} e y_i de tal maneira a satisfazer o princípio do vizinho mais próximo é o valor médio definido como

$$p_{i-1} = \frac{y_{i-1} + y_i}{2}. \tag{2.7}$$

2.3.2 Encontrar o melhor codebook para uma partição

O próximo passo para otimizar o projeto de codebooks consiste em resolver o segundo problema: encontrar o melhor codebook para uma dada partição.

Para resolver este problema, utiliza-se a condição de centróide. Esta condição afirma que, para uma região R_i, o valor ótimo de saída y_i é "centro de massa"da região R_i. Matematicamente, a propriedade do centro de massa é definido como

$$y_i = E\big[X|X \in R_i\big]. \tag{2.8}$$

Ou seja, para uma partição R_i, o melhor valor de saída y_i que minimiza o erro é justamente o valor esperado dado a distribuição de probabilidade da fonte dado que o valor está dentro da partição R_i.

Para provar a condição do centróide, deve-se voltar para o cálculo do erro D. Este erro é calculado como

$$D = \sum_{i=1}^{N} \int_{R_i} (x - y_i)^2 f_x(x) dx. \tag{2.9}$$

Considera-se, então, a análise de uma região específica R_i, ou seja,

$$D_i = \int_{R_i} (x - y_i)^2 f_x(x) dx. \tag{2.10}$$

A Equação acima pode ser expressa da seguinte forma:

$$P_j \int_{-\infty}^{\infty} (x - y_j)^2 f_{X|X \in R_j}(x) dx. \tag{2.11}$$

A integral da Equação 2.11 pode ser expressa através de uma probabilidade condicional resultando em

$$P_j E\big[(x - y_i)^2 | X \in R_j\big]. \tag{2.12}$$

O objetivo é minimizar o valor descrito na Equação 2.12 e uma opção consiste em utilizar uma propriedade de minimização descrito em [16] que informa que o valor α que minimiza a função $E\big[(Y - \alpha)^2 | Y \in G\big]$ é $\alpha = E\big(Y|Y \in G\big)$. Aplicando este princípio para minimizar a Equação 2.12 tem-se o seguinte:

$$\begin{aligned} y_i &= \int_{R_i} x f_{X|R_i}(x) dx, \\ y_i &= \frac{\int_{R_i} x f_X(x) dx}{\int_{R_i} f_X(x) dx}. \end{aligned} \tag{2.13}$$

Além do quantizador escalar descrito anteriormente, destaca-se o quantizador vetorial que é bastante utilizado. Este quantizador é descrito a seguir.

Este método de otimização é utilizado para projetar os codebooks deste trabalho. Iniciando de um codebook inicial, resolve-se o problem de encontrar a melhor partição. Em seguida, dada a melhor partição encontrada, resolve-se o primeiro problema encontrando o melhor codebook. O processo se repete até minimizar o erro D.

2.4 Quantização vetorial

A quantização vetorial (VQ) pode ser visualizada como uma generalização da quantização escalar no sentido de uma ou mais amostras podem ser quantizadas por vez. Como exemplo, considere uma sequência x de 12 elementos: $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$. Para quantizar esta sequência utilizando VQ, primeiro é necessário definir a dimensão do vetor de quantização. Para este exemplo, o vetor de três dimensões será utilizado, ou seja, a sequência original x será dividida em quatro sequências de três elementos $s_1 = [1, 2, 3]$, $s_2 = [4, 5, 6]$, $s_3 = [7, 8, 9]$ e $s_4 = [10, 11, 12]$.

Após definida a dimensão do vetor, o próximo passo é definir o codebook a ser utilizado. Cada elemento do codebook C consiste de uma sequência de três elementos. Considere, para fins didáticos, que o codebook consiste nas seguintes codewords:

1. $c_1 = [1, 2.6, 2.8]$
2. $c_2 = [10, 5, 6]$
3. $c_3 = [30, 40, 50]$
4. $c_4 = [20, 15, 10]$
5. $c_5 = [40, 45, 50]$
6. $c_6 = [4.5, 5.2, 6.1]$
7. $c_7 = [7.1, 8.1, 9.1]$
8. $c_8 = [10.2, 11.2, 12.2]$

Portanto, para quantizar cada uma das sequências s_1, s_2 e s_3, basta encontrar, para cada sequência s_i, qual *codeword* c_j minimiza o erro e entre a sequência s_i e a sequência quantizada c_i. Para este exemplo, o erro é definido na Equação 2.14 seguir:

$$e = (s_i - C_i)^2. \tag{2.14}$$

Neste exemplo, para codificar um codebook contendo c_n elementos, são necessários b bits. Neste caso, como o codebook contém oito sequências ($C_{size} = 8$), apenas três bits são necessários, ou seja,

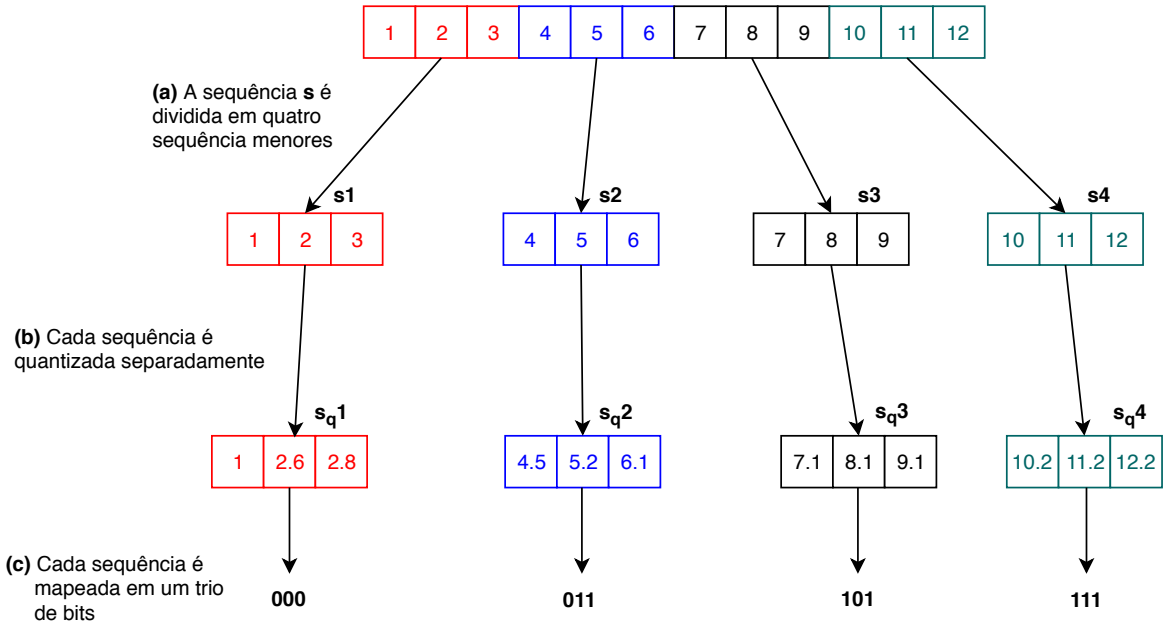
$$\begin{aligned} b &= log_2\big(C_{size}\big), \\ b &= log_2\big(8\big), \\ b &= 3. \end{aligned} \tag{2.15}$$

Para calcular a taxa de bits a ser transmitida, basta dividir o número de bits transmitidos pelo total de amostras. É importante ressaltar que o número de bits B transmitidos é igual ao número de bits b necessário para quantizar cada sequência s_i vezes o número de sequências a serem quantizadas (4 sequências). Ou seja,

$$\begin{aligned} r &= \frac{4 \times 3}{2}, \\ r &= \frac{12}{12}, \\ r &= 1. \end{aligned} \tag{2.16}$$

A Figura 2.7 abaixo ilustra graficamente a quantização vetorial descrita anteriormente. A sequência original s é dividida em quatro sequências: s_1, s_2, s_3 e s_4 (passo (a) da Figura 2.7). Então, cada sequência é quantizada separadamente (passo **b** da Figura 2.7). Finalizando, cada sequência quantizada é mapeada em três bits (pois o codebook c é composto de oito sequências diferentes).

Figura 2.7: Exemplo de quantização vetorial.



Fonte: o autor.

2.5 Trabalhos relacionados

Nesta seção serão abordadas importantes técnicas utilizadas para comprimir o sinal transmitido no *fronthaul*. Listam-se brevemente os métodos a seguir:

2.5.1 *Linear Prediction Coding*

Esta técnica é baseada no cálculo da predição linear e foi proposta em [19]. A n-ésima amostra $\hat{x}[n]$ pode ser predita utilizando P amostras anteriores devidamente escalados pelo preditor de ordem P. A Equação 2.17 apresenta o cálculo da predição para um preditor de ordem P, ou seja,

$$\hat{x}[n] = P_0 x[n-1] + P_1 x[n-2] + P_3 x[n-3] + ... + P_i x[n-i]. \tag{2.17}$$

O LPC é utilizado para predizer a n-ésima amostra dentro de um símbolo OFDM. Seja essa amostra a ser transmitida representado pela variável $x[n]$. Então, o primeiro passo consiste

em enviar para o decoder as amostras $x[0]$ até $x[P-1]$ (Isto é visualizado através da chave que está na posição 1 na Figura 2.17). Isto é feito para iniciar a memória do preditor de ordem P. Após isto, a chave da Figura 2.17 é mudada para a posição 2 e cada amostra $x[n]$ é predita utilizando a Equação 2.17.

Após predizer o valor $x[n]$, o erro de predição entre o valor predito e valor original é quantizado utilizando quantização escalar. Após a quantização escalar, o sinal $I[n]$ é comprimido utilizando o algoritmo de Huffman.

Um fator importante no cálculo do preditor diz respeito a sua ordem. Preditores de ordem mais alta tendem a oferecer mais robustez ao sistema. Entretanto, aumentar a ordem do preditor indefinidamente não é vantajoso pois a partir de uma certa ordem, o ganho de predição não aumenta [19].

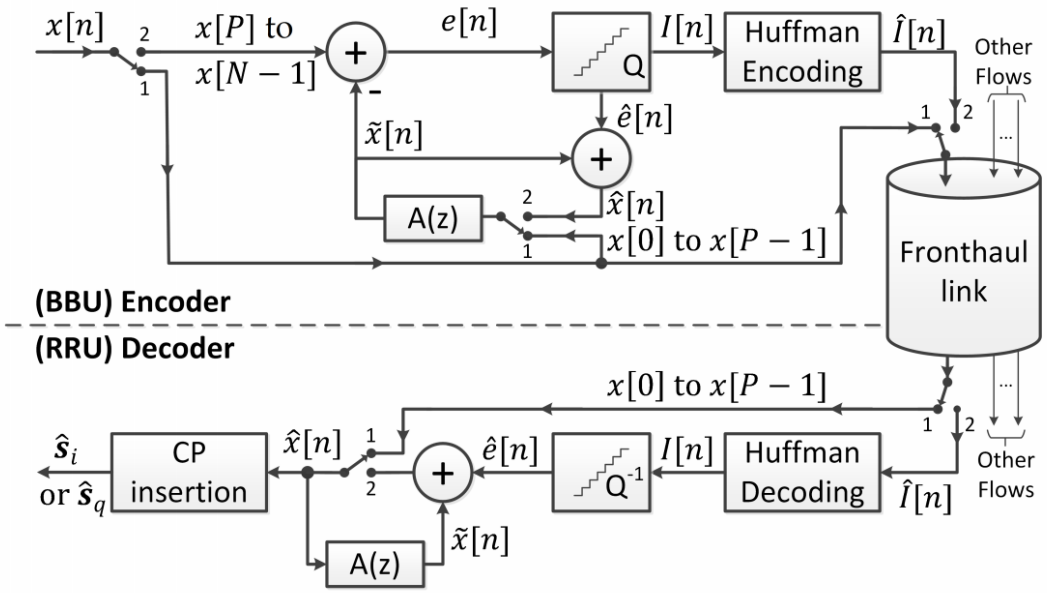
Uma característica importante deste sistema está no seu custo computacional. Como mostrado em [19], o número de operações aritméticas necessárias para quantizar um valor complexo é descrito como

$$C_p = \frac{2(P+2)(N-P)}{N+N_{CP}} \tag{2.18}$$

na qual P é a ordem do preditor, N é a quantidade de amostras sem prefíxo cíclico (igual ao tamanho da *Inverse Fast Fourier Transform*) e N_{CP} é igual ao número de amostras com a adição do prefixo cíclico.

Outra vantagem do uso desta técnica diz respeito a sua latência pois uma amostra $x[n]$ é processada e enviada para o decoder em um tempo reduzido pois, como preditores de ordem alta não beneficiam o sistema, é aconselhável utilizar preditores de ordem baixa, reduzindo a latência a um tempo bem curto.

Figura 2.8: Sistema de compressão baseada na predição linear.

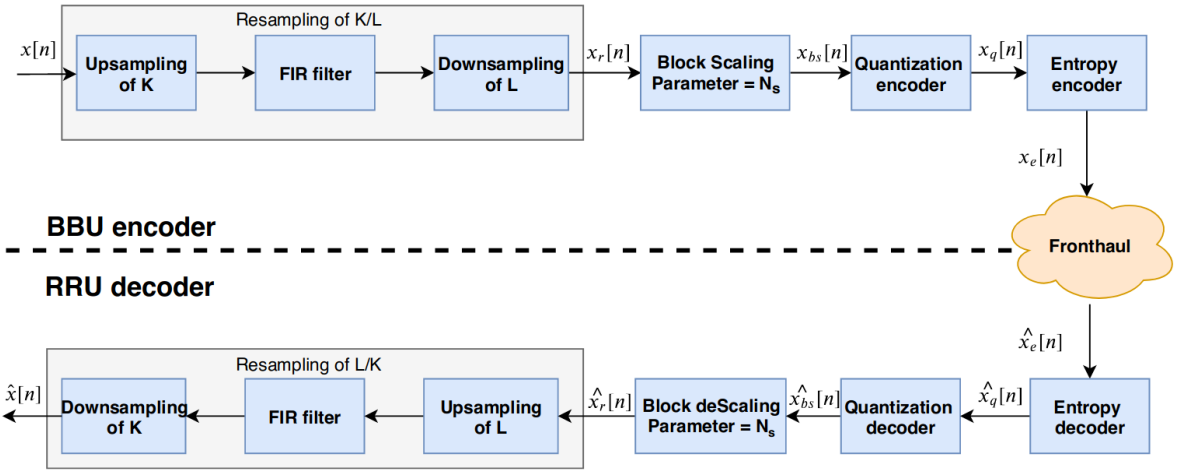


Fonte: [19].

2.5.2 Compressão baseada em *resampling*

Nesta subseção será explicado em detalhes todos os blocos envolvidos na compressão demonstrada em [20] e em [21]. A Figura 2.9 apresenta o diagrama de blocos geral utilizado no trabalhos [20] e [21].

Figura 2.9: Diagrama blocos da técnica baseada em resampling.



Fonte: o autor.

Na Figura 2.9, o primeiro bloco consiste na operação conhecida como *resampling*. Este grande bloco consiste de três blocos menores nomeados a seguir: o bloco de *upsampling*, o filtro FIR e o bloco de *downsampling*.

O bloco de *upsampling* é responsável por realizar o *upsampling* do sinal por um fator K. Esta operação pode ser realizada através da inserção de zeros entre os elementos de uma sequência [22]. Como exemplo, considere a sequência $u[n] = [1, 2, 3, 4, 5]$. Ao realizar o upsampling desta sequência por um fator $K = 2$, temos, como resultado, a seguinte sequência $u_{up}[n] = [1, 0, 0, 2, 0, 0, 3, 0, 0, 4, 0, 0, 5]$.

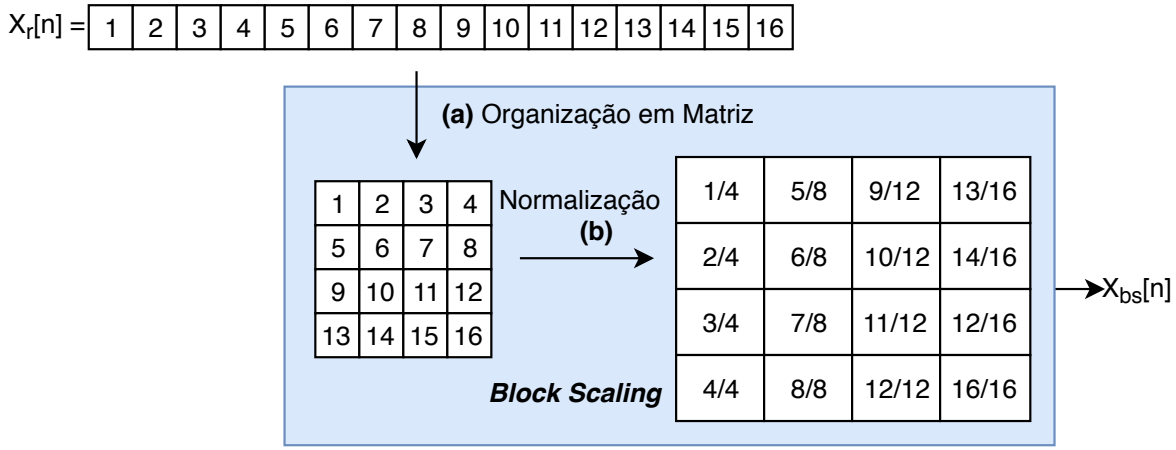
Um fator importante deste *upsampling* consiste no espelhamento do espectro do sinal, pois realizar o *upsampling* resulta na réplica do espectro no domínio da frequência. Portanto, é necessário atenuar essas réplicas e para isto aplica-se o segundo bloco do sistema que consiste no filtro FIR (*Finite Impulse Response*).

O terceiro bloco da operação de resampling consiste na operação de *downsampling* de L. Esta operação é feita através da remoção a cada $L - 1$ elementos (esta operação mantém o primeiro elemento intacto). Como exemplo, considere a sequência $u_f[n] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$. Realizar uma operação de *downsampling* de ordem 3 resulta na seguinte sequência $u_{ds}[n] = [1, 4, 7, 10]$.

Todas estas operações (*upsampling* de K, filtragem e *downsampling* de L) resultam na sequência $x_r[n]$. Esta sequência é então utilizada como entrada para o bloco conhecido como *Block Scaling* (BS). Este bloco reorganiza o sinal $x_u[n]$ em uma matriz com Ns linhas e S/Ns colunas em que a variável S é igual ao tamanho do vetor de entrada $x_r[n]$. Em seguida, cada linha é normalizada pelo seu valor máximo. A Figura 2.10 ilustra o algoritmo do *Block Scaling*.

No exemplo da Figura 2.10, o vetor de entrada é $x_r[n]$ que consiste de 16 elementos, logo o tamanho do vetor é igual a $S = 16$. Neste caso específico, o vetor é organizado em uma matriz com $Ns = 4$ linhas e $\frac{S=16}{Ns=4} = 4$ colunas. Os primeiros 4 elementos são postos na primeira linha; o próximos 4 elementos na segunda linha e assim por diante. Em seguida, cada linha é normalizada pelo seu valor máximo, sendo o valor máximo da primeira linha igual a 4; o valor máximo da segunda linha igual a 8; o valor máximo da terceira linha igual a 12 e o valor máximo da quarta linha igual a 16.

Figura 2.10: Ilustração do *Block Scaling*.



Fonte: o autor.

Após esta operação, o sinal resultante é denominado de $x_{bs}[n]$. Este sinal é utilizado como entrada para o bloco de quantização. É importante ressaltar que cada coluna da matriz $x_bs[n]$ é quantizada por vez, resultando no sinal $x_q[n]$. O trabalho descrito em [20] utiliza quantização escalar enquanto o trabalho descrito em [21] utiliza quantização vetorial neste bloco.

Finalizando o bloco do encoder, o sinal é comprimido utilizando o algoritmo de Huffman [23] resultando no sinal que será transmitido pelo *fronthaul*.

O bloco de decoder consiste em todas as respectivas operações inversas resultando no sinal $\hat{x}[n]$.

Assim como o trabalho descrito em [20] e em [21], este trabalho também adiciona, como inovação, um sistema de quantização baseada no *resampling*, *block scaling*, quantização e código de Huffman. A diferença é que este trabalho utiliza a *Trellis Coded Quantization* no encoder e decoder da Figura 2.9. O TCQ é uma técnica de quantização baseada no uso de treliças específicas. No próximo capítulo apresenta-se em detalhes o funcionamento do TCQ assim como um exemplo numérico é apresentado em detalhes. Finalizando o próximo capítulo, o custo computacional do TCQ é apresentado em detalhes.

Capítulo 3

TRELLIS CODED QUANTIZATION

Neste capítulo será apresentado em detalhes o funcionamento e a estrutura da *Trellis Coded Quantization* (TCQ). Por ser parte importante da inovação deste trabalho, será dedicado um capítulo inteiro para detalhar o funcionamento do TCQ. O TCQ foi proposto tendo como base a técnica conhecida como *Trellis Coded Modulation* (TCM). As duas técnicas estão relacionadas de tal maneira que um sistema é similar ao outro sendo que a diferença entre eles é que o TCM é uma técnica utilizada na codificação e modulação de sinais enquanto o TCQ é utilizado na quantização de sequências.

Pelo fato do TCM ter surgido primeiro que o TCQ, este sistema será apresentado primeiro. Após sua explicação, o entendimento do TCQ será facilitado.

3.1 *Trellis Coded Modulation*

Trellis Coded Modulation (TCM) é uma técnica que combina modulação e codificação com o objetivo de ser utilizado na transmissão digital de sinais. O primeiro artigo que descreve o TCM foi apresentado em 1976 e tem como título *"On improving data-link performance by increasing the channel alphabet and introducing sequence coding"* [24]. Entretanto, somente com a publicação do artigo intitulado *"Channel coding with multilevel/phase signals"* [25] que o TCM começou a ganhar peso na área da pesquisa.

3.2 Ganho de código do TCM

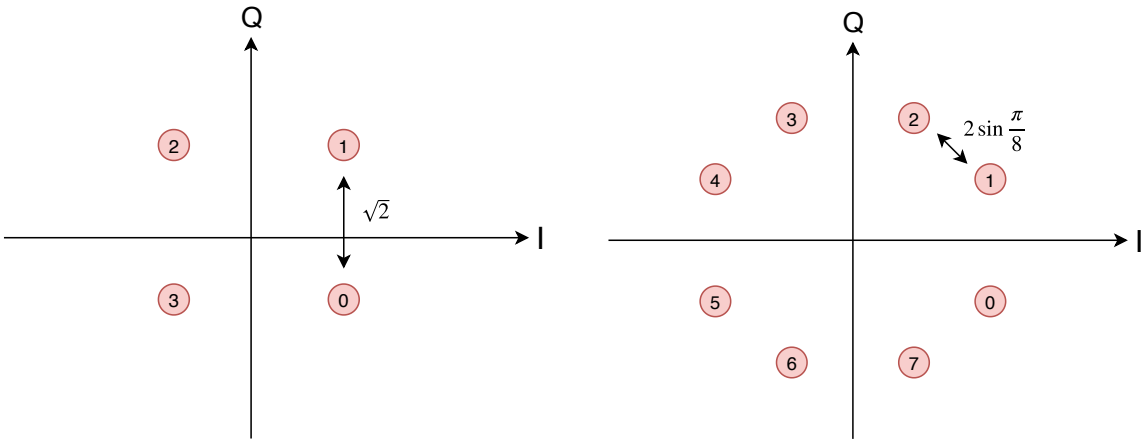
O principal fator motivante do TCM consiste no ganho de código que é obtido em relação ao sistema sem codificação. O ganho de código é calculado encontrando a mínima distância δ_0 entre duas sequências codificadas e mínima distância δ_1 entre duas sequências do sistema original não codificado. Então, para expressar o ganho de código G entre as distâncias δ_0 e δ_1 aplica-se a divisão a seguir:

$$G = \log_2 \frac{\delta_0}{\delta_1}. \tag{3.1}$$

Para análise de ganho de código, será comparado o desempenho do sistema de transmissão utilizando TCM em relação ao sistema não codificado que seria utilizado normalmente. A ideia

central do TCM consiste na expansão da modulação, ou seja, caso nosso sistema não codificado utilize r-PSK como modulação, o TCM correspondente utilizará $(2r)$-PSK. Esta expansão tem como objetivo aumentar a mínima distância entre duas sequências possíveis. Entretanto, como mostra as Figuras 3.2(a) e 3.2(b), apenas optar por aumentar a modulação de 4-PSK para 8-PSK resulta na diminuição da mínima distância possível entre duas sequências. Para aumentar-la, necessita-se limitar conjunto de sequências possíveis que podem ser transmitidas na modulação 8-PSK e o TCM realiza isto através de uma treliça específica que modela este conjunto de tal maneira que as mínima distâncias δ_0 entre duas sequências do TCM ainda será maior que no caso da modulação não codificada 4-PSK, resultando em ganho de código.

Figura 3.1: Modulações 4-PSK e 8-PSK.



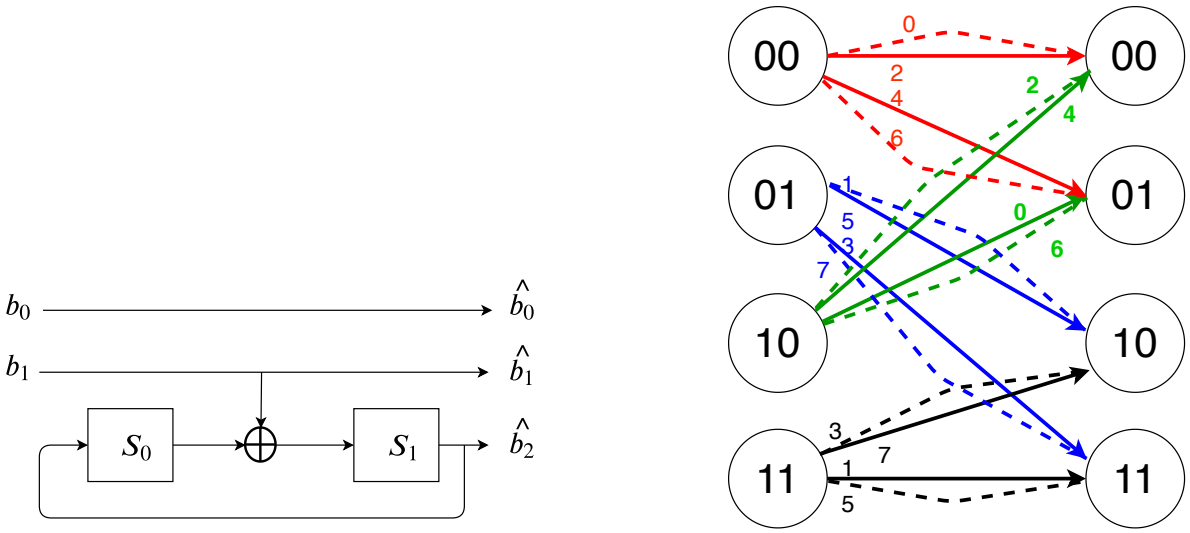
(a) 4-PSK (b) 8-PSK

Fonte: o autor.

3.3 Aplicando Treliças para aumentar a mínima distância

Para aumentar a mínima distância possível entre duas sequências, o TCM utiliza treliças específicas desenvolvidas por Ungerboeck. Esta treliça proposta está exemplificada na Figura 3.3(b) abaixo. Para encontrar a menor distância entre duas sequências utilizando TCM, é importante analisar primeiro a máquina de código convolucional que aplica a treliça apresentada na Figura 3.3(a). Nota-se que há dois conjuntos de bits: os bits codificados que são utilizados como entrada no código convolucional e o bit não codificado que não é utilizado como entrada para o código convolucional. A seguir, será analisado o ganho de código que é obtido ao utilizar-se o TCM.

Figura 3.2: Código convolucional e treliça correspondente proposto por Ungerboeck.



(a) Código Convolucional do TCM

(b) Treliça correspondente.

Fonte: o autor.

3.4 Ganho de código do TCM

A treliça correspondente do TCM é visualizada na Figura 3.3(b) acima e respectivo código convolucional é apresentado na Figura 3.3(a). Os estados do código convolucional são representados pelas variáveis S_0, S_1, S_2 e S_3 e podem assumir os valores 0 e 1 totalizando quatro possíveis estados (00, 01, 10 e 11) dependendo dos bits de entrada e do estado atual dos valores S_0 e S_1.

Como mostra o código convolucional da Figura 3.3(a) acima, cada par de bit utilizado como entrada para o código convolucional (b_0 e b_1) resulta na saída três bits ($\hat{b_0}$, $\hat{b_1}$ e $\hat{b_2}$). Destes três bits, o bit b_0 não possui relação com o código convolucional. Este bit é chamado de bit não codificado. O bit b_1, por ser utilizado na codificação e resultar na saída nos bits $\hat{b_1}$ e $\hat{b_2}$, é chamado de bit codificado.

A treliça apresentada na Figura 3.3(b) possui, para cada saída, dois caminhos paralelos ilustrados pelas linhas tracejadas. Estes caminhos existem pois, para cada bit codificado resultando em um par de bits, há um bit não codificado que pode assumir os valores 0 e 1. Ou seja, para cada saída do código convolucional, adiciona-se os bits 0 e 1 criando dois caminhos paralelos.

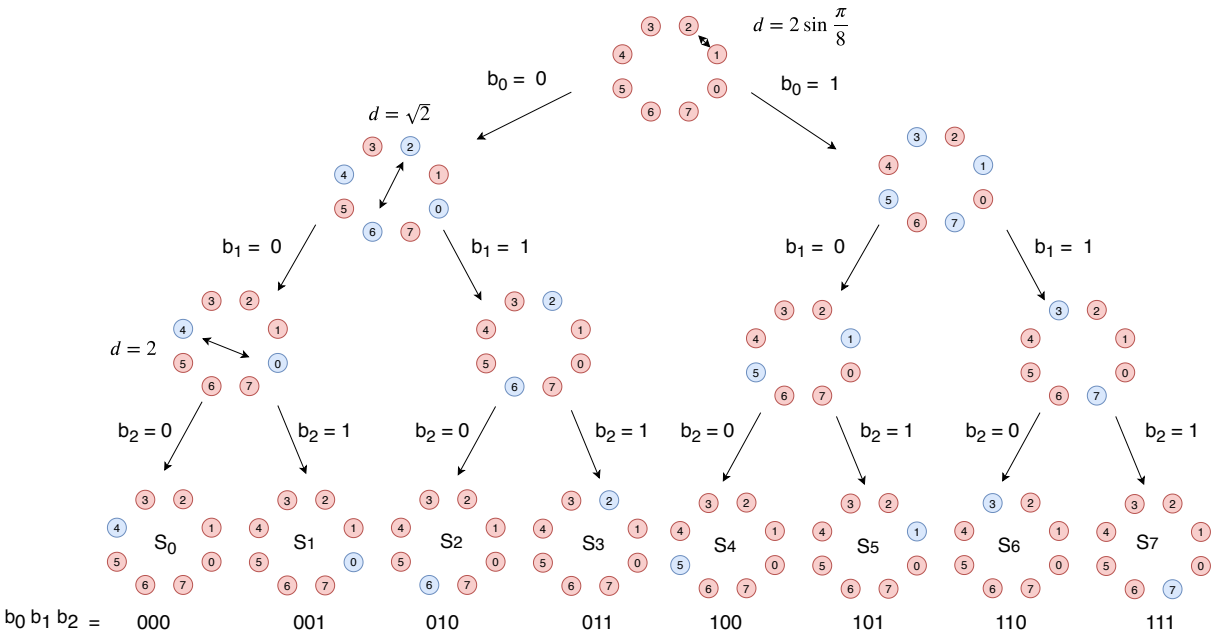
3.5 Particionamento da constelação

Um aspecto extremamente importante do projeto do TCM consiste no particionamento da constelação. Este particionamento é responsável por realizar o mapeamento entre os bits de saída e o símbolo 8-PSK correspondente. Um mapeamento realizado de maneira errada

proporciona um conjunto totalmente diferente de sequências permitidas do conjunto S o que pode resultar em uma diminuição da distância mínima entre duas sequências, diminuindo o ganho de código do sistema.

A forma de particionamento proposta por Ungerboeck [26], [27] consiste em separar, a cada iteração do algoritmo, cada conjunto de símbolos é dividido em dois novos subconjuntos disjuntos de tal maneira que a mínima distância entre cada símbolo do subconjutno seja maximizada. A Figura 3.3 ilustra este processo de divisão iterativa. Para cada divisão, dois novos subconjuntos são gerados e para cada subconjunto é atribuído o bit 0 e 1 como mostra a Figura 3.3. Ao final do processo de particionamento, tem-se o mapeamento entre os três bits de saída e o seu respectivo símbolo. Por exemplo, aos bits 000 é atribuído o símbolo S_0, aos bits 001 é atribuído o símbolo S_1 e assim por diante.

Figura 3.3: Particionamento Ungerboeck da modulação 8-PSK.

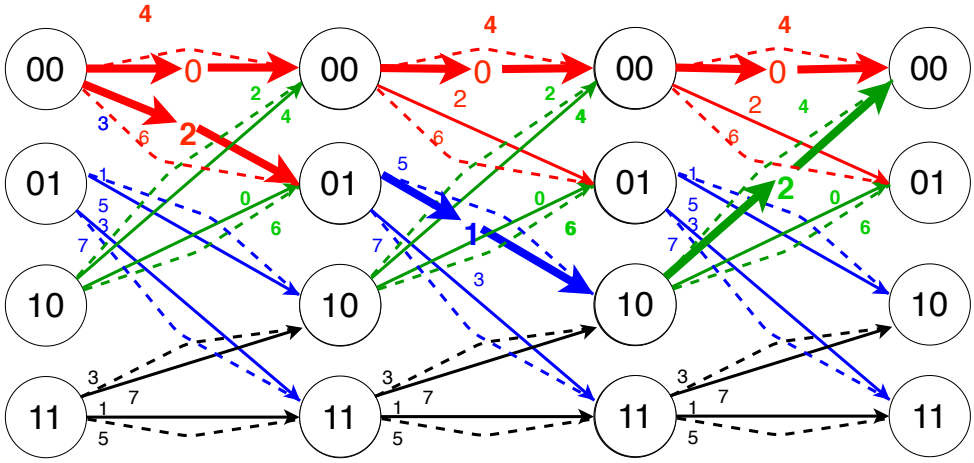


Fonte: o autor.

Para encontrar o ganho de código deste sistema, é necessário encontrar duas sequências que possuem a menor distância euclidiana entre elas. Para encontra-las, um método é partir do estado 00, percorrer a treliça e retornar imediatamente para o estado 00. Este caminho mínimo é ilustrado na Figura 3.4 abaixo. Os números ao redor de cada caminho representam o símbolo 8-PSK mapeado na saída da constelação.

Para calcular a distância mínima, basta somar cada uma das distâncias individuais. Na treliça da Figura 3.4, cada número ao redor de cada caminho representa o símbolo 8-PSK correspondente da Figura 3.3. Caso as duas sequências sejam $SEQ_1 = seq_{1a}$, seq_{1b}, seq_{1c} e $SEQ_2 = seq_{2a}$, seq_{2b} e seq_{2c} Calcula-se a distância entra cada símbolo individualmente $(seq_{1a} - seq_{2a})^2$ + $(seq_{1b} - seq_{2b})^2$ + $(seq_{1c} - seq_{2c})^2$. Neste exemplo específico, $SEQ_1 = 000$

Figura 3.4: Busca do caminho mínimo entre duas sequências.



Fonte: o autor.

e $SEQ_2 = 212$. A Equação 3.2 apresenta o cálculo da distância D_1.

$$\begin{aligned} D_1 &= (\sqrt{2})^2 + (0.765)^2 + (\sqrt{2})^2, \\ D_1 &= 2 + 0.586 + 2, \\ D_1 &= 4.586. \end{aligned} \tag{3.2}$$

Entretanto, há outra distância que deve ser considerada : os caminhos paralelos. Ao observar a treliça cuidadosamente, nota-se que cada caminho paralelo possui uma distância mínima $D_2 = 2$. Como exemplo, o caminho paralelo 000 e 100 como ilustrado na Figura 3.5, devido a divisão da modulação demonstrada anteriormente, estão separados por uma distância $D_2 = 2^2$. Todos os caminhos paralelos possuem esta distância.

Finalizando, a mínima distância encontrada entre duas sequência é o mínimo valor entre D_1 e D_2, ou seja,

$$\begin{aligned} D_{min} &= min(4.586, 4), \\ D_{min} &= 4. \end{aligned} \tag{3.3}$$

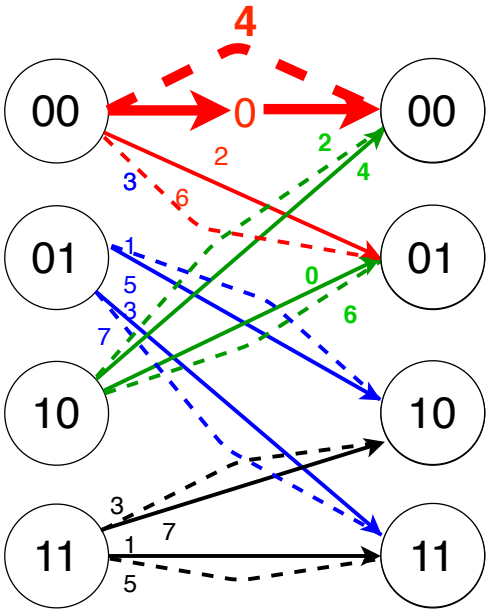
Para encontrar o ganho de código entre as duas sequências em decibéis, basta aplicar o logaritmo na base 2 na razão entre as duas unidades como mostra a Equação 3.4:

$$\begin{aligned} G &= \log_2 \frac{4}{2}, \\ G &= 3dB. \end{aligned} \tag{3.4}$$

Nota-se que houve um ganho de 3 dB sem a necessidade de consumir recursos do canal de transmissão. Para aumentar este ganho, pode-se utilizar treliças com uma quantidade maior de estados mas esta abordagem tem, como consequência, o aumento do custo computacional do sistema.

Após esta análide de ganho de código do TCM, será explicada como funciona o transmissor e o receptor do TCM. A Figura 3.6 ilustra o sistema de transmissão e recepção sendo que o transmissor utiliza um código convolucional de taxa 1/2 para gerar, a cada dois bits de entrada,

Figura 3.5: Caminho mínimo considerando os caminhos paralelos.



Fonte: o autor.

três bits de saída. Cada trio de bit é mapeado em um símbolo da constelação 8-PSK. No receptor, para cada sequência recebida, é aplicado o algoritmo de Viterbi na treliça da Figura 3.6 para encontrar, dentre todas as possíveis sequências permitidas, a sequência que minimiza a distância euclidiana em relação a sequência recebida.

Figura 3.6: Particionamento Ungerboeck da modulação 8-PSK.

Fonte: o autor.

3.6 TCM: exemplo numérico

Para fins didáticos, nesta seção será mostrada um exemplo numérico do uso do TCM. Neste exemplo, dois bits por amostra serão inseridos por vez na entrada do código convolucional de taxa $1/2$ e, portanto, para cada dois bits de entrada, três bits de saída serão gerados. Esses três bits de saída serão utilizados pelo *mapper* para gerar o símbolo correspondente da modulação

8-ASK.

Antes de iniciar o exemplo, os seguintes parâmetros precisam ser definidos:

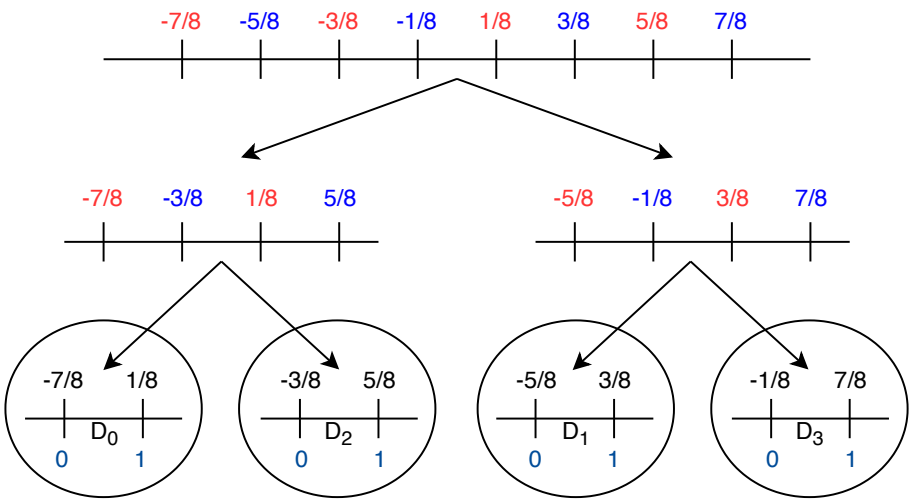
- **Definição do código convolucional:** o código convolucional a ser utilizado neste exemplo é mostrado na Figura 3.8(b). Ele contém quatro estados e, para cada dois bits inseridos na entrada, três bits são gerados na saída. O código convolucional foi configurado para iniciar no estado 00 como mostra a figura abaixo.

- **Definição da modulação :** neste exemplo será utilizado a modulação 8-ASK e, como neste caso será utilizado uma fonte de distribuição de probabilidades uniforme, os seguintes símbolos foram escolhidos: -0.875, -0.625, -0.375, -0.125, 0.125, 0.375, 0.625 e 0.875.

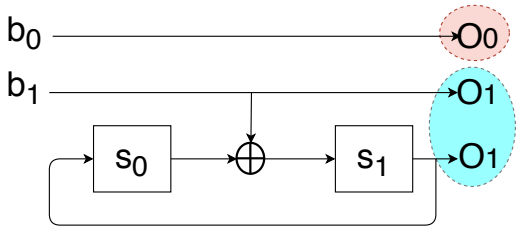
- **Realizar o particionamento de Ungerboeck da modulação 8-ASK:** o particionamento tem como objetivo realizar o mapeamento entre os bits de saída do código convolucional e o símbolo escolhido. O mapeamento é ilustrado na Figura 3.8(a) abaixo. Este mapeamento organiza os oito símbolos em quatro *cosets* D_0, D_1, D_2 e D_3. Os três bits de saída do código convolucional são responsáveis por selecionar o símbolo escolhido sendo que os bits O_1 e O_2 (região azul da Figura 3.8(b)) selecionam um dos quatro *cosets* e o bit O_o (região vermelha) seleciona o elemento dentro do *coset* escolhido.

Após esta configuração, basta definir a sequência de entrada a ser processada. Neste caso, a sequência será um grupo de par de bits definidos como s = 00, 01, 10. Com estes parâmetros devidamente configurados, basta iniciar o exemplo. Neste caso, o exemplo será dividido em iterações, sendo que cada iteração representa um par de bits processados.

Figura 3.7: Estado do código convolucional antes e depois de processar o bit 0 na primeira iteração.



(a) Estado inicial do código convolucional



(b) Saída e estado final do código convolucional

Fonte: o autor.

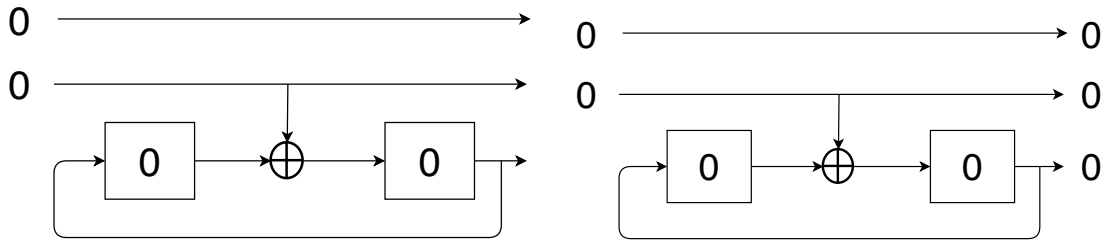
3.6.1 Transmissor TCM: primeira iteração

Na primeira iteração, insere-se o primeiro par de bits (00) da sequência s no código convolucional (Figura 3.9(a)). Como o estado do código é 00 e os bits 00 são inseridos na entrada, os três bits de saída serão 000 e o código convolucional move-se do estado 00 para o estado 00 (ou seja, não houve alteração). Como os bits de saída são 000 (Figura 3.9(b)), o *mapper* seleciona o símbolo -0.875.

3.6.2 Transmissor TCM: segunda iteração

Na segunda iteração, o estado inicial do código convolucional ainda está no estado 00. Desta vez, os bits de entrada serão 01 (Figura 3.10(a)). Como o código convolucional está no estado 00 e os bits inseridos na entrada são 01, os bits de saída são 010 (Figura 3.10(b)). Desta

Figura 3.8: Estado do código convolucional antes e depois de processar o bit 0 na primeira iteração.



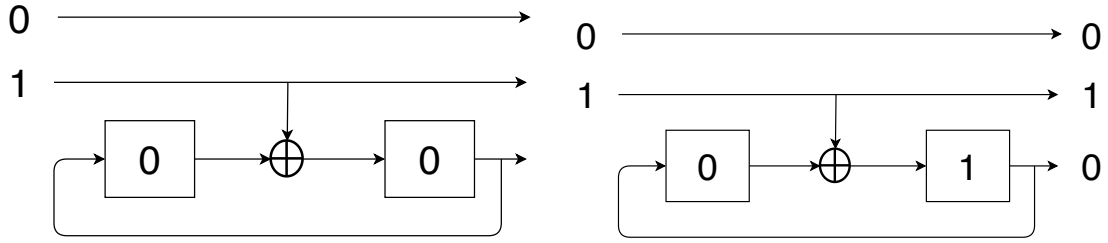
(a) Estado inicial do código convolucional (b) Saída e estado final do código convolucional

Fonte: o autor.

vez, houve alteração do estado do código convolucional que migrou do estado 00 para o estado 01.

O *mapper* recebe como entrada os bits 010 e retorna como saída o símbolo correspondente: -0.375.

Figura 3.9: Estado do código convolucional antes e depois de processar o bit 0 na primeira iteração.



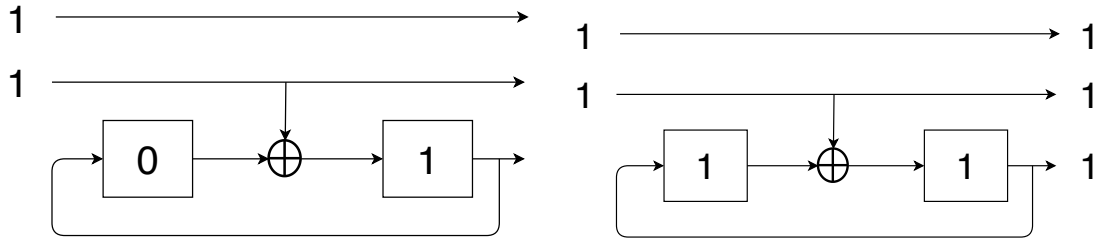
(a) Estado inicial do código convolucional (b) Saída e estado final do código convolucional

Fonte: o autor.

3.6.3 Transmissor TCM: terceira iteração

Na terceira iteração, o estado inicial do código convolucional é 01 e os bits de entrada são 11 (Figura 3.11(a)). Os bits de saída do código são 111 e o estado final do código convolucional é 11 (Figura 3.11(b)). O *mapper* então seleciona o símbolo 0.875.

Figura 3.10: Estado do código convolucional antes e depois de processar o bit 0 na primeira iteração.



(a) Estado inicial do código convolucional (b) Saída e estado final do código convolucional

Fonte: o autor.

3.7 Receptor TCM: algoritmo de Viterbi

O receptor do TCM utiliza-se do algoritmo de Viterbi para encontrar, dentre todas as possíveis sequências, aquela que minimiza o erro quadrático. Neste exemplo, a sequência transmitida $x[n] = -0.875, -0.375, 0.875$ sofrerá os efeitos do ruído no canal. O receptor determina a sequência a ser escolhida através do algoritmo de Viterbi. O algoritmo é o mesmo ilustrado na seção do encoder do TCQ explicado em grandes detalhes na seção 3.9.

3.8 Do TCM para o TCQ

Os criadores do TCQ conseguiram aproveitar grande parte da estrutura e funcionamento do TCM ao implementar a técnica de quantização conhecida como TCQ. A seguir, ilustra-se os principais componentes do TCQ bem como um exemplo numérico é mostrado.

3.8.1 Adição da redundância

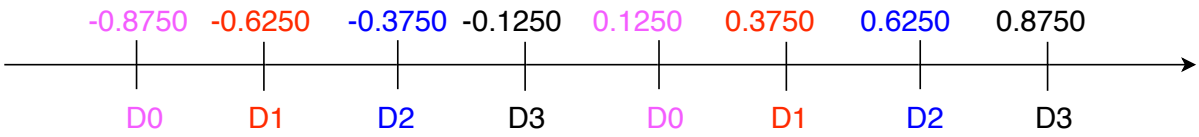
Relembrando a Seção 3.2, a redundância do TCM é adicionada com o objetivo de aumentar a robustez do sistema contra ruído e isto é feito dobrando o número de constelações. Por exemplo, se o sistema não codificado utiliza 4-PSK, o TCM utilizará 8-PSK. Entretanto, simplesmente dobrar o número de constelações diminui a distância entre os pontos. Portanto, é necessário gerenciar a escolha dos símbolos de tal maneira que a distância entre as sequências seja maximizada. o TCM faz esse gerenciamento através do particionamento da constelação mostrado na Seção 3.5 e através do uso de treliças específicas.

Como o TCQ adiciona esta redundância? No lugar de dobrar o número de constelações, o TCQ dobra o número de codewords. Por exemplo, caso um quantizador utilize 4 codewords (2 bits), no TCQ, no número de codewords torna-se 8 (3 bits). Entretanto, assim como o TCM aumenta a redundância sem aumentar os recursos utilizados no canal, o TCQ dobra o número de codewords sem aumentar o número de bits transmitidos no canal. Como isto é feito? Através da mesma treliça utilizada no TCM, ou seja, o TCQ gerencia quais codewords podem ser escolhidas a cada iteração da treliça.

Outro fator que aumenta a redundância no TCM é o particionamento da constelação. Através deste particionamento, é criado o *mapping* entre os bits de entrada e seu respectivo símbolo da constelação. O TCQ realiza o mesmo particionamento só que, ao invés de modulações (4-PSK, 8-PSK, 16-PSK), o TCQ particiona o codebook projetado. A seguir, será explicado brevemente como o TCQ implementa estas redundâncias.

Por exemplo, considere um projetista que deseje implementar um quantizador de 2 bits utilizando TCQ. Para isto, seja a fonte S de distribuição de probabilidade uniforme com valores entre -1 e 1. Como mostrado em [28], um codebook que minimiza a relação SQNR consiste nos seguintes valores apresentados na Figura 3.11 abaixo. Este particionamento é o mesmo relatado no exemplo do TCM ilustrado na Figura 3.8(a):

Figura 3.11: Ilustração do codebook utilizado no exemplo bem como a organização dos codebooks em quatro *cosets*.



Fonte: o autor.

Como é possível notar, o mesmo sistema de particionamento é utilizado para organizar as codewords em *cosests* D_0, D_1, D_2 e D_3. Cada *coset* contém exatamente b codewords.

Após a expansão do codebook C e do seu particionamento, o TCQ está preparado para ser implementado. A seguir, será apresentado em detalhes o processo de quantização de uma sequência utilizando TCQ.

3.9 TCQ: exemplo numérico

Para melhor entendimento, nesta seção será apresentado um exemplo de quantização utilizando TCQ. Neste exemplo, o TCQ será utilizado com a treliça de quatro estados e a seguinte sequência S será quantizada: $-0.7162, -0.1565, 0.8315$.

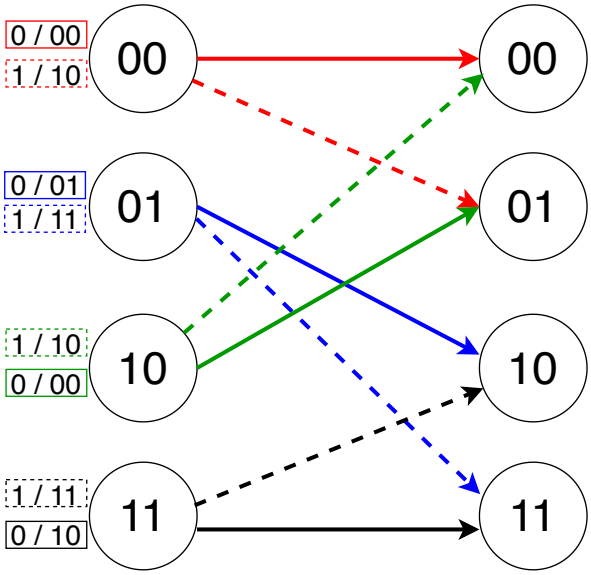
Será utilizado a mesma fonte S de distribuição uniforme descrita na Seção 3.8.1 acima.

Neste exemplo, o quantizador de 2 bits será utilizado. Através da expansão do número de codewords (Neste caso, de 4 para 8 codewords) e do particionamento de Ungerboeck do codebook C descrito na seção anterior, o codebook é organizado em quatro *cosets* D_0, D_1, D_2 e D_3 a seguir:

$$\begin{aligned} D0 &= [-0.8750 + 0.1250], \\ D1 &= [-0.6250 - 0.3750], \\ D2 &= [-0.3750 + 0.6250], \\ D3 &= [-0.1250 + 0.8750]. \end{aligned} \tag{3.5}$$

Com os elementos de cada conjunto devidamente preenchidos, o próximo passo demonstrado aqui será a escolha do uso da treliça. Neste caso, a mesma treliça proposta por Ungerboeck para o TCM será utilizado. Esta treliça é ilustrada na Figura 3.12 abaixo:

Figura 3.12: Treliça a ser utilizada no exemplo. O bit a esquerda de cada retângulo representa o bit de entrada do código convolucional enquanto os dois bits a direita representam os bits de saída do código convolucional.



Fonte: o autor.

Com todos estes parâmetros previamente definidos, a quantização pode ser iniciada.

Na primeira iteração do algoritmo, o primeiro elemento da sequência $S(1) = -0.7162$ será quantizado utilizando um quantizador escalar simples para cada um dos conjuntos $D1, D2, D3$ e $D4$ resultando nos seguintes elementos quantizados: $-0.8750, -0.3750, +0.1250, +0.6250$. Cada quantização resulta nos seguintes erros de quantização E_1, E_2, E_3 e E_4 a seguir:

$$\begin{aligned} E_0 &= \big[-0.7162 - (-0.8750)\big]^2 = 0.02522, \\ E_1 &= \big[-0.7162 - (-0.3750)\big]^2 = 0.00832, \\ E_2 &= \big[-0.7162 - (+0.1250)\big]^2 = 0.11642, \\ E_3 &= \big[-0.7162 - (+0.6250)\big]^2 = 0.34952. \end{aligned} \tag{3.6}$$

Esses erros foram encontrados através da diferença quadrática entre o valor original e o valor quantizado, ou seja,

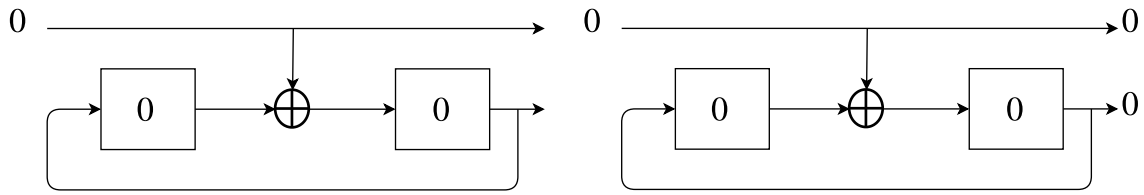
$$E = (x - xq)^2. \tag{3.7}$$

3.9.1 Primeira iteração

Inicia-se o percurso inserindo o bit 0 como entrada do código convolucional. Como o estado inicial do código convolucional é 00 e ele recebe como entrada o bit 0, logo a saída serão os bits 00. A Figura 3.14(a) ilustra o estado inicial do código convolucional; a Figura 3.14(b) ilustra o estado final do código convolucional e a Figura 3.14(c) ilustra o caminho percorrido

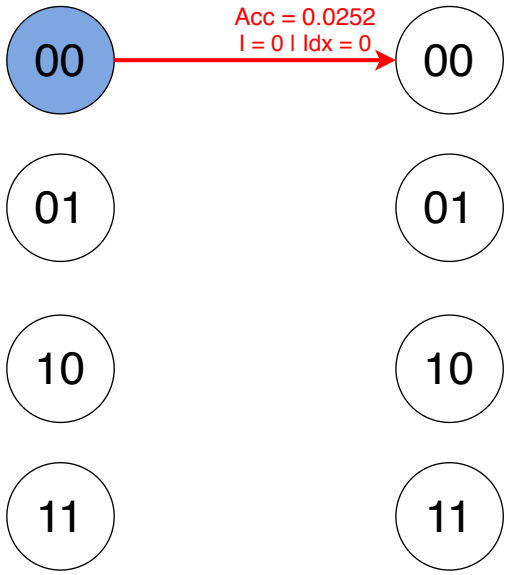
na treliça. Como os bits de saída do código são 00, seleciona-se o *coset* $D0$. Então, o elemento mais próximo de $D0$ da variável de entrada é selecionado (-0.8750) como é mostrado na Figura 3.14(d). Calcula-se o erro acumulado que é igual a $E = 0 + E_0 = 0.02522$. Por ter migrado do estado 00 para o estado 00, não houve alteração dos valores da máquina de estado do código convolucional que continou no estado 00. Então, salva-se o bit de entrada do código convolucional representado pela variável I na Figura 3.14(c) e a variável *index bit* responsável por referenciar o valor -0.8750 representado pela variável Idx na Figura 3.14(c).

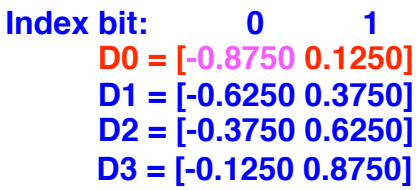
Figura 3.13: Estado do código convolucional antes e depois de processar o bit 0 na primeira iteração.



(a) Estado inicial do código convolucional

(b) Saída e estado final do código





(c) Estado da treliça utilizada.

(d) Lista dos cosets.

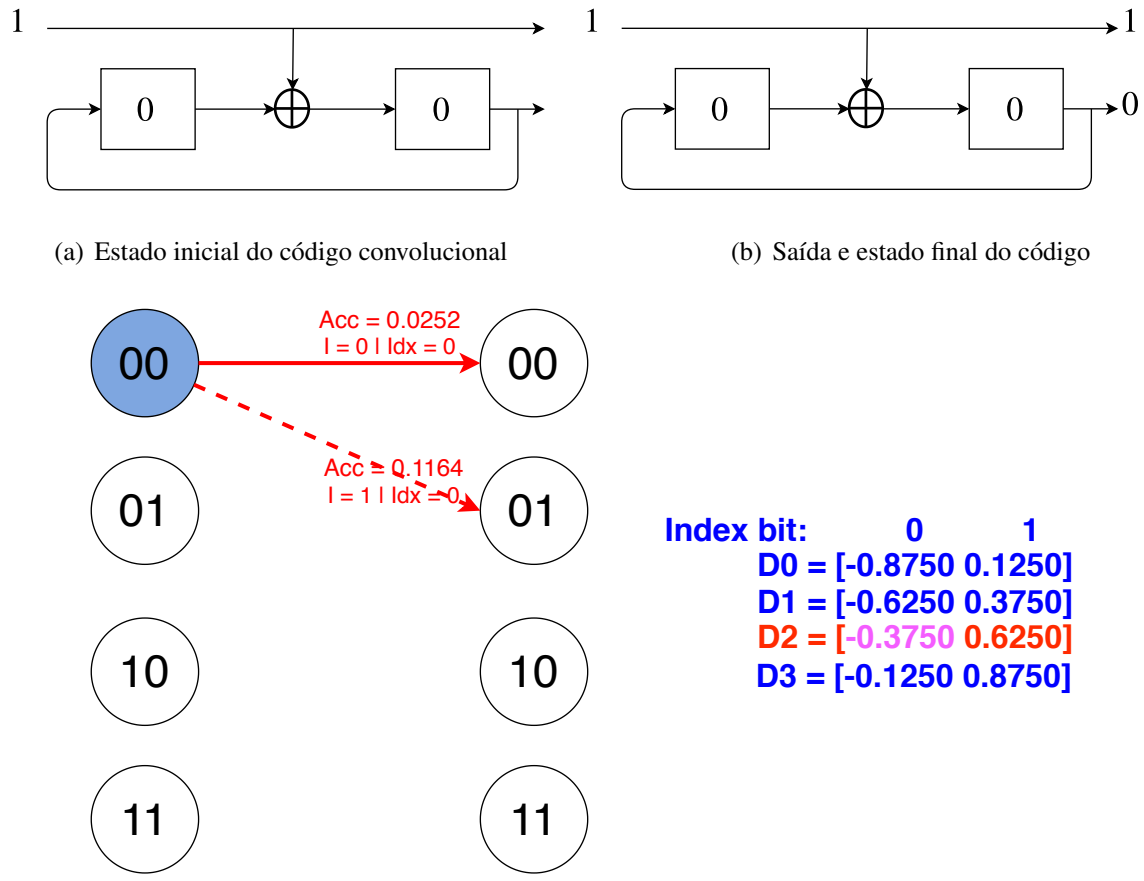
Fonte: o autor.

3.9.2 Segunda Iteração

Na próxima iteração do algoritmo, o estado do código convolucional continua como 00. É importante salientar neste caso que o valor 00 é o estado inicial do código e não deve ser confundido com o estado final (que também é 00) apresentado na iteração anterior. Desta vez, será inserido o bit 1 na entrada do código (Figura 3.15(a)) o que resulta, como saída, os bits 10

(Figura 3.15(b)). Estes bits de saída são responsáveis pela escolha do *coset* $D2$. Seleciona-se o elemento dentro do coset $D2$ mais próximo de $S(1) = -0.7162$ que é -0.3750 como mostra a Figura 3.15(d). O *index bit* representado por este elemento é 0 e o *coset bit* (bit utilizado como entrada do código convolucional) é 1. Estes bits são salvos. O último valor a ser armazenado é o erro acumulado que é igual a $E = 0 + E_1$ $E = 0 + 0.11642$ $E = 0.11642$.

Figura 3.14: Ilustração da segunda iteração do TCQ.



(a) Estado inicial do código convolucional

(b) Saída e estado final do código

(c) Estado da treliça utilizada.

(d) Lista dos cosets.

Fonte: o autor.

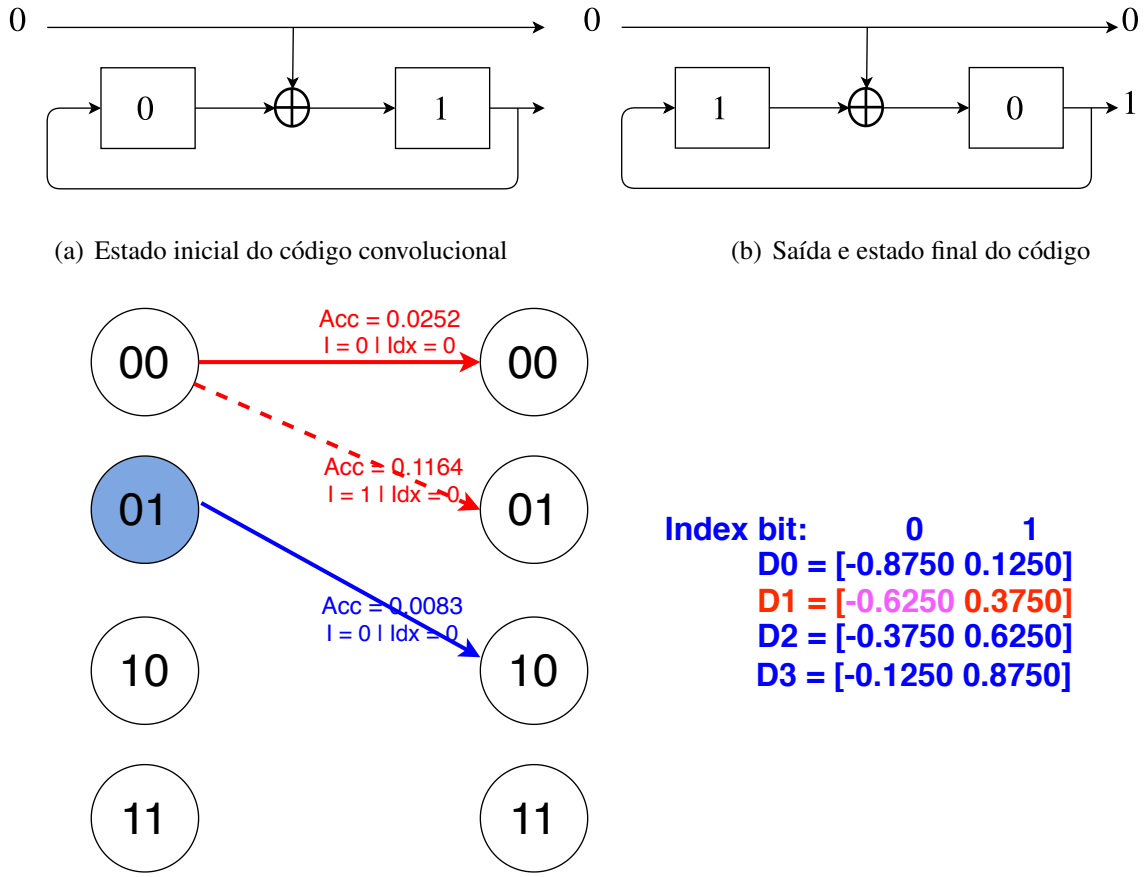
3.9.3 Terceira Iteração

Nesta terceira iteração, o estado do código convolucional é alterado para 01. Então, o bit 0 é aplicado como entrada para o código convolucional (Figura 3.16(a)) e a saída resultante são os bits 01 (Figura 3.16(b)). Estes bits de saída são responsáveis pela escolha do coset $D1$. O próximo passo é selecionar, dentro deste coset, o elemento que é mais próximo de $S(1) = -0.7162$ (Figura 3.16(d)). Esta operação já foi realizada através da quantização ilustrada no início do capítulo e, portanto, o elemento dentro do coset $D1$ escolhido é -0.6250.

Com este elemento escolhido, calcula-se o erro quadrático acumulado que é igual ao valor $0 + 0.00832 = 0.00832$.

O último passo desta iteração é salvar o *coset bit* 0 e o *index bit* 0.

Figura 3.15: Estado do código convolucional antes e depois de processar o bit 0 na terceira iteração.



(a) Estado inicial do código convolucional

(b) Saída e estado final do código

(c) Estado da treliça utilizada.

(d) Lista dos cosets.

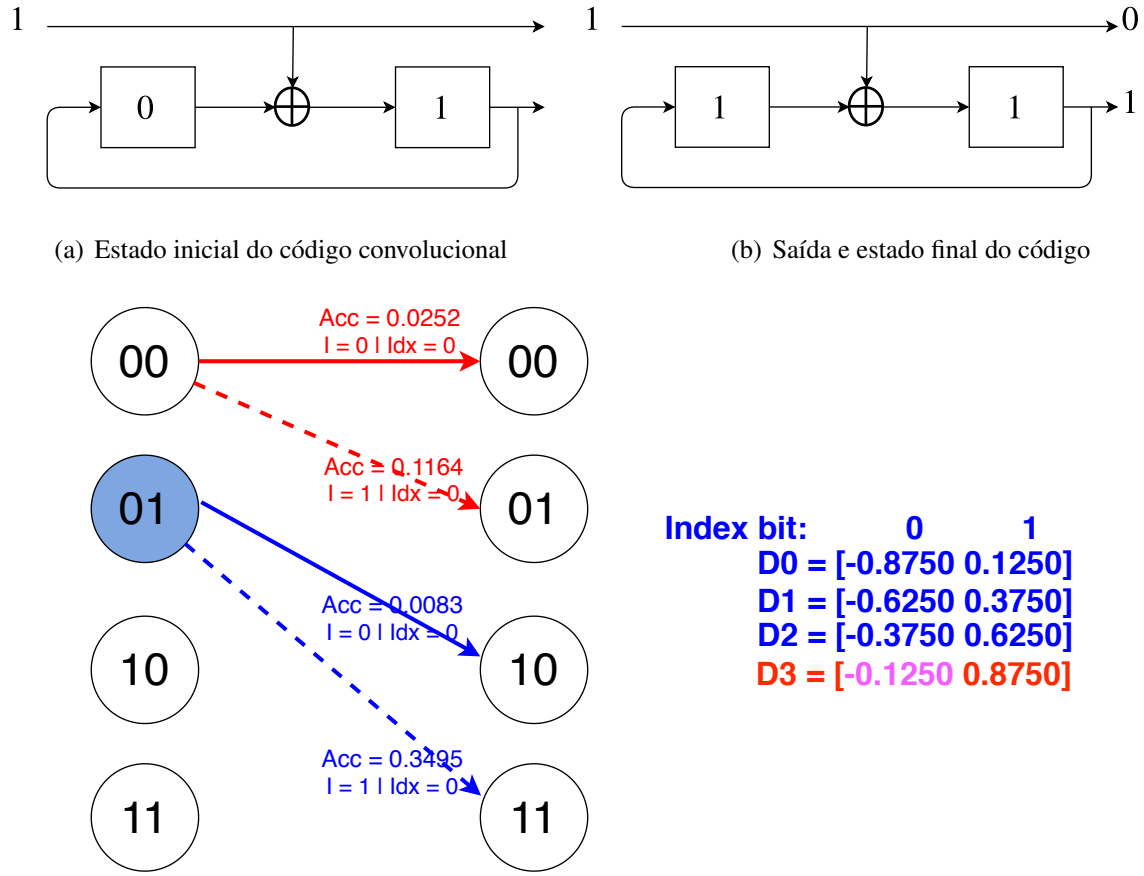
Fonte: o autor.

3.9.4 Quarta Iteração

A quarta iteração é similar a terceira iteração. A única diferença é o bit utilizado como entrada no código convolucional. Desta vez, o bit 1 é utilizado como entrada (Figura 3.17(a)). Aplicar o bit 1 como entrada ao código convolucional resulta como saída os bits 11 (Figura 3.17(b)). Estes bits de saída são responsáveis pela escolha do coset $D3$. Escolhe-se, dentro deste *coset* $D3$, o elemento mais próximo de $S(1) = -0.7162$ que é -0.1250 (Figura 3.17(d)). É importante lembrar que esta escolha já foi feita com a quantização realizada no início do capítulo.

Então, calcula-se o erro acumulado que, neste caso, por não haver nenhum erro acumulado previamente, ele é igual ao valor $E_3 = 0.34952$.

Figura 3.16: Estado do código convolucional antes e depois de processar o bit 1 na quarta iteração.



(a) Estado inicial do código convolucional

(b) Saída e estado final do código

(c) Estado da treliça utilizada.

(d) Lista dos cosets.

Fonte: o autor.

3.9.5 Quinta iteração

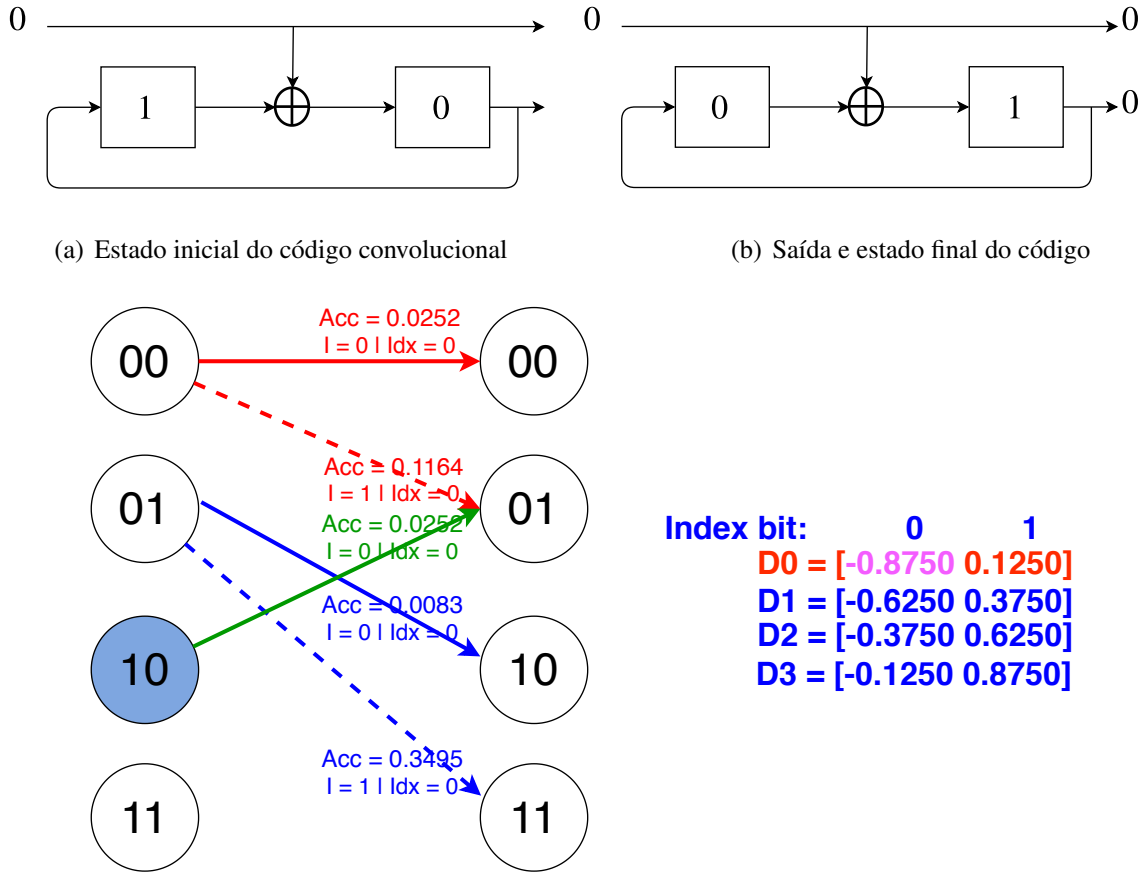
Na quinta iteração do algoritmo, o estado inicial do código convolucional e da treliça é 10. Nesta etapa, o bit 0 é utilizado como entrada para o código convolucional (Figura 3.18(a)). Esta entrada resulta como saída o valor 00 (Figura 3.18(b)). Esta saída é utilizada então para selecionar o *coset* $D0$. Então, seleciona-se o elemento do coset $D0$ que é mais próximo do valor $S(1) = -0.7162$ (Figura 3.18(d)). Este valor, portanto, é -0.8750.

Após a selação do elemento, obtem-se o erro acumulado que igual a $0 + E_0 = 0 + 0.02522 = 0.02522$.

3.9.6 Sexta iteração

Na sexta iteração do algoritmo de quantização, o estado inicial da treliça é 10 (o mesmo estado da iteração anterior). A única diferença nesta etapa é que o bit 1 será utilizado como entrada para o código convolucional (Figura 3.19(a)). Inserir o bit 1 como entrada no código

Figura 3.17: Estado do código convolucional antes e depois de processar o bit 0 na quinta iteração.



(a) Estado inicial do código convolucional

(b) Saída e estado final do código

(c) Estado da treliça utilizada.

(d) Lista dos cosets.

Fonte: o autor.

convolucional resulta como saída os bits 10 (Figura 3.19(b)). Estes bits de saída referenciam o *coset* $D2$. Então, seleciona-se o elemento do *coset* $D2$ que mais se aproxima do valor $S(1) = -0.7162$ que é -0.3750 (Figura 3.19(d)). Lembrando novamente que esta operação já foi realizada anteriormente e, portanto, nã há custos adicionais agora.

Após selecionar o valor -0.3750, calcula-se o erro acumulado. Como esta é a primeira iteração, ela é igual a diferença quadrática $E_2 = 0.11642$.

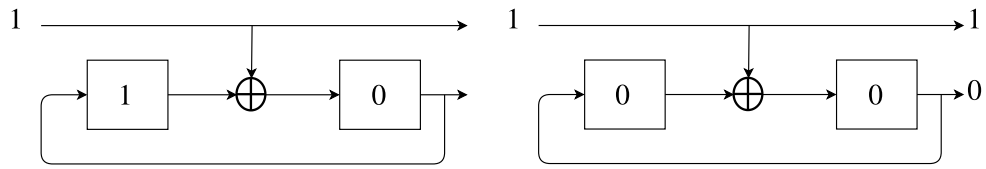
Salva-se novamente o *subset bit* 1 e o index bit 0 como é mostrado na Figura 3.19(c).

3.9.7 Sétima iteração

Na sétima iteração, o estado do código convolucional é 11. Neste passo, o bit 0 é inserido como entrada para o código convolucional (Figura 3.20(a)) resultando como saída os bits 01 (Figura 3.20(b)). Não há alteração no estado do código convolucional nesta etapa.

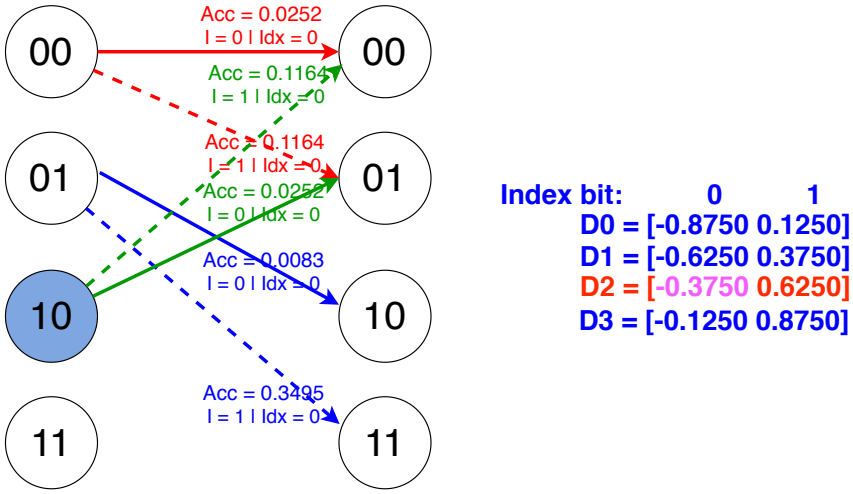
Como os bits de saída do código convolucional são 01 o *coset* $D1$ é selecionado. Dentro

Figura 3.18: Estado do código convolucional antes e depois de processar o bit 1 na sexta iteração.



(a) Estado inicial do código convolucional

(b) Saída e estado final do código



(c) Estado da treliça utilizada.

(d) Lista dos cosets.

Fonte: o autor.

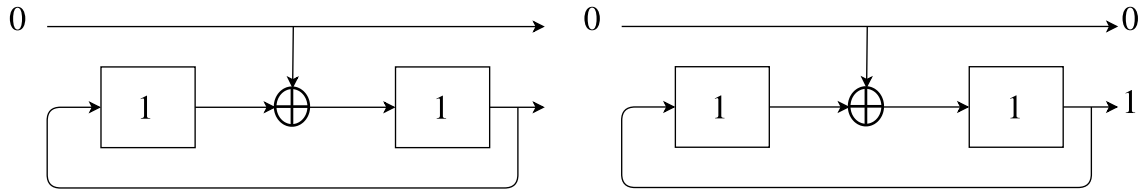
do *coset* $D1$, o elemento que mais se aproxima de $S(1) = -0.7162$ é -0.6250 (Figura 3.20(d)). Calcula-se o erro acumulado que é igual a diferença quadrática entre o valor original e o valor quantizado acrescentado do erro no estado anterior. Como esta é a primeira amostra ainda a ser quantizada, o erro acumulado nesta etapa é igual a $0 + E_1 = 0.00832$.

3.9.8 Oitava iteração

Esta iteração é peculiar pois ela finaliza o percurso da treliça para a primeira amostra. Nesta iteração, o valor dos estados do código convolucional é o mesmo da iteração anterior (11). A diferença, desta vez, é que o bit 1 é utilizado como entrada para o código convolucional (Figura 3.21(a)) resultando, como saída, os bits 11 (Figura 3.21(b)). Seleciona-se então o *coset* $D3$ mais próximo da variável de entrada (Figura 3.21(d)). O valor escolhido é -0.1250 e o novo estado do código convolucional é 10 (Figura 3.21(b)). Calcula-se então o erro acumulado que é igual a $0 + E_3 = 0 + 0.34952$.

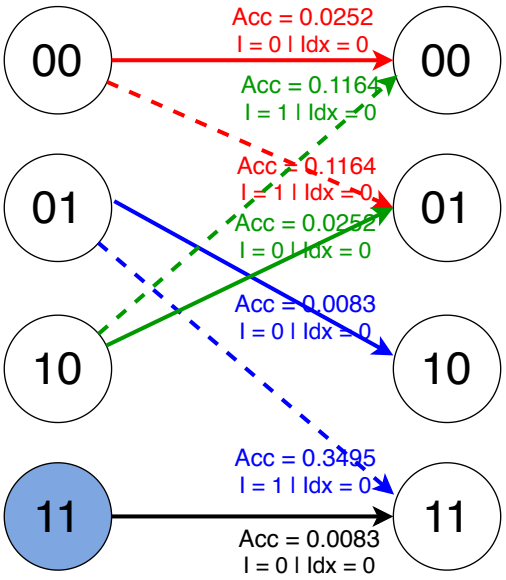
Esta iteração é finalizada salvando os correspondentes *subset bit* (1) e o *index bit* (0) como é mostrado na Figura 3.21(c).

Figura 3.19: Estado do código convolucional antes e depois de processar o bit 0 na sétima iteração.

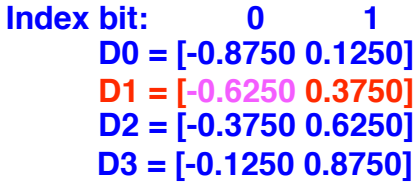


(a) Estado inicial do código convolucional

(b) Saída e estado final do código



(c) Estado da treliça utilizada.



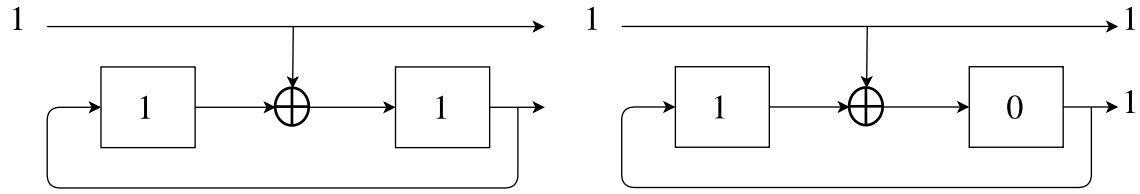
(d) Lista dos cosets.

Fonte: o autor.

3.9.9 Nona iteração: exclusão de caminhos

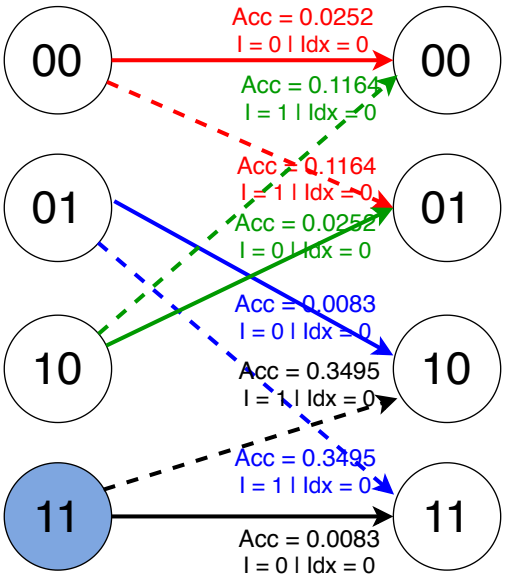
Após cada uma das oito iterações apresentadas anteriormente, tem-se a seguinte configuração apresentada na Figura 3.22(a). Nota-se que, antes de partir para a segunda amostra a ser quantizada $S(2) = -0.1565$, em cada estado da treliça a dois caminhos chegando e dois erros acumulados. Necessita-se, portanto, escolher qual dos dois caminhos deverá ser eliminado para que somente um erro acumulado seja utilizado. Esta operação é bastante simples pois, em cada estado, mantém-se o caminho com o menor erro acumulado e elimina-se o caminho com o maior erro acumulado. A Figura 3.22(b) apresenta a treliça já com os caminhos desnecessários eliminados. Após esta exclusão, pode-se seguir para a quantização da próxima amostra da sequência.

Figura 3.20: Estado do código convolucional antes e depois de processar o bit 1 na oitava iteração.

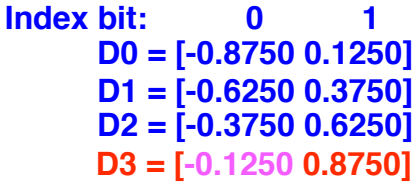


(a) Estado inicial do código convolucional

(b) Saída e estado final do código



(c) Estado da treliça utilizada.



(d) Lista dos cosets.

Fonte: o autor.

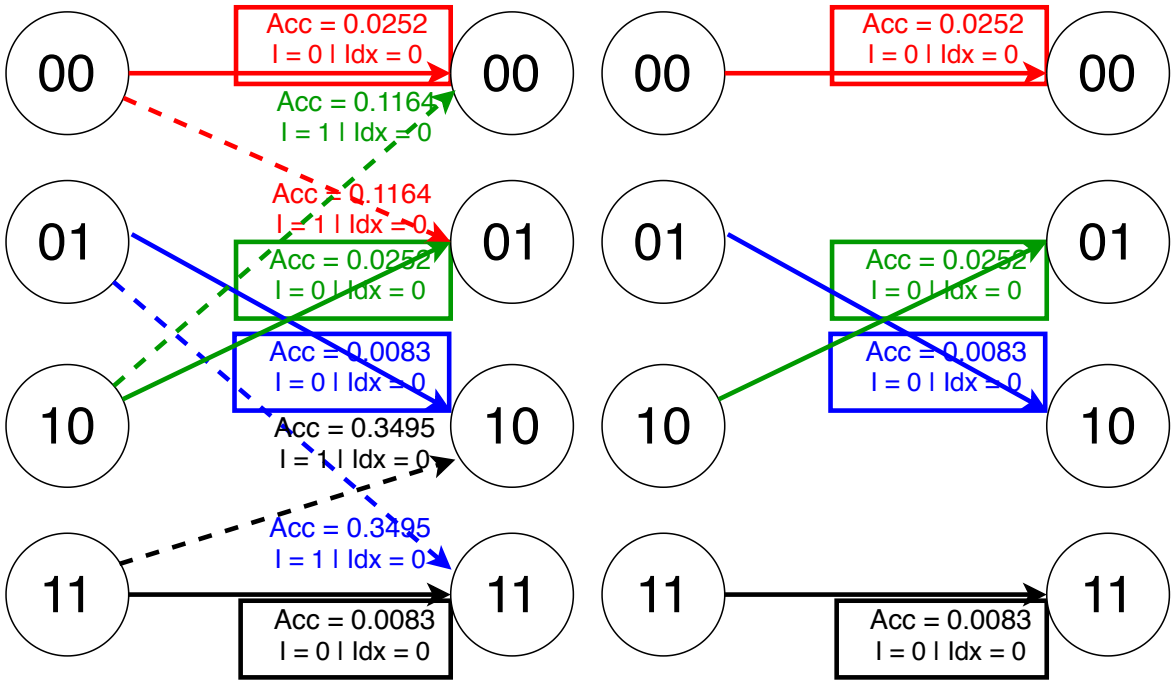
3.9.10 Segunda amostra

Nesta etapa, não serão apresentados em profundos detalhes os oito passos necessários para processar a segunda etapa da treliça pois eles são extremamente similares as oito etapas anteriores. Portanto, a Figura 3.22 apresenta já o algoritmo após estes oitos estados.

3.9.11 Eliminação dos caminhos

Após as oito etapas utilizando a segunda amostra, deve-se eliminar novamente os caminhos desnecessários (Figura 3.24(a)). O processo é exatamente o mesmo do que foi explicado anteriormente (mantem-se os caminhos com o menor erro acumulado e elimina-se os caminhos com o maior erro acumulado para cada estado). A Figura 3.24(b) apresenta o estado da treliça após esta eliminação.

Figura 3.21: Exclusão dos caminhos.



(a) Selação dos maiores erros acumulados (b) Exclusão dos maiores erros acumulados

Fonte: o autor.

3.9.12 Terceira amostra

A última amostra a ser processada neste exemplo é $S(3) = 0.8315$. Como explicado anteriormente, as oito iterações desta amostra são examente a mesma das etapas anteriores e, portanto, será apresentado apenas o resultado final.

Após as oito iterações, os caminhos desnecessários são excluídos como é mostrado na Figura 3.24 que é o estado final da treliça.

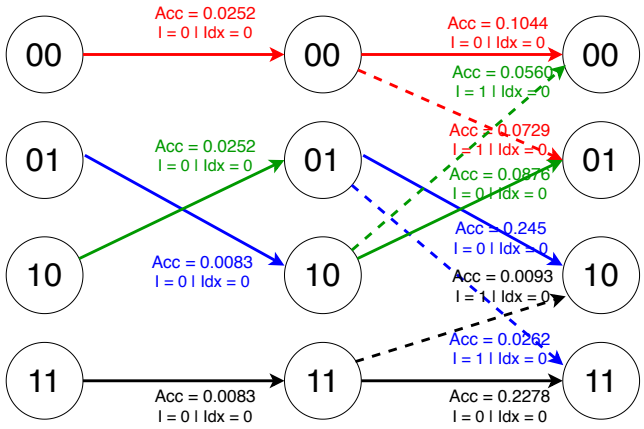
3.9.13 Quantização: escolha dos bits

Todas estas etapas foram necessárias para escolher o caminho dentro das treliças que minimiza o erro quadrático. O primeiro passo é escolher o menor erro acumulado no final da treliça. Como é mostrado na Figura 3.25, o menor erro acumulado é 0.0281. Então, percorre-se a treliça de trás para frente sempre coletando os *coset bits* e os *index bits* para cada amostra resultando nos seguintes bits que serão utilizados como saída do encoder como é mostrado na Equação 3.8 abaixo:

$$bits = 11, 10, 00. \tag{3.8}$$

É importante salientar que esta sequência de bits representa a sequência invertida pois o caminho foi percorrido da última amostra para a primeira amostra sendo necessário, portanto,

Figura 3.22: Treliça final para as duas primeiras amostras.



Fonte: o autor.

apenas inverter a sequencia de pares de bits para se obter a sequência final, ou seja, os valores de bits são iguais a

$$bits = 00, 10, 11. \tag{3.9}$$

A última etapa do decoder é adicionar ao vetor de bits o estado inicial da treliça. Como a treliça foi percorrida do último elemento para o primeiro e o primeiro elemento contem o estado 10. Estes dois bits serão adicionados resultando no vetor de bits como é mostrado a seguir:

$$bits = 10, 00, 10, 11. \tag{3.10}$$

Estes bits serão transmitidos até o decoder que será explicado a seguir.

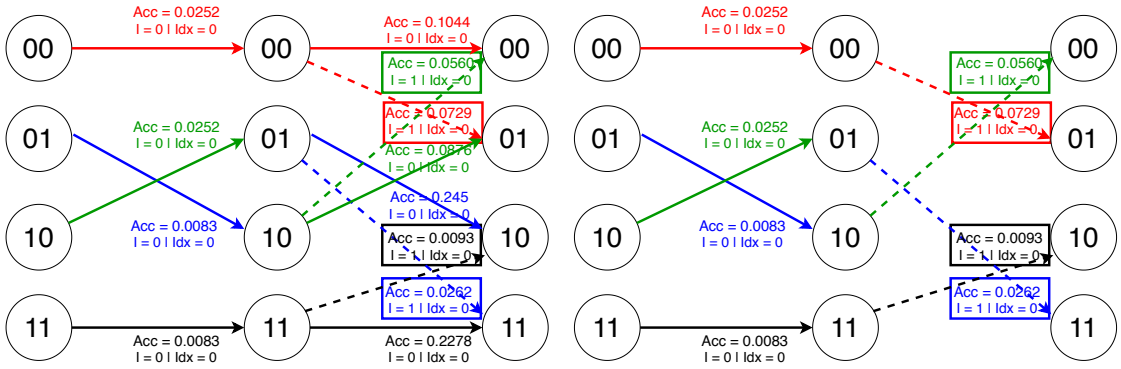
3.9.14 TCQ decoder

O decoder é bem mais simples que o encoder consistindo basicamente de um código convolucional e um *table look up*. O primeiro passo é utilizar os dois primeiros bits da sequência recebida como estado inicial do código convolucional. Neste caso, a sequência é $10, 00, 10, 11$. Como os dois primeiros bits da sequência são 1 e 0, o estado inicial S_0 do código convolucional será configurado como 1 e o estado inicial S_1 do código convolucional como 0. O próximo passo após receber os bits do enconder é separa-los entre os *coset bits* e os *index bits* como mostra a Figura 3.26 abaixo:

Antes de prosseguir, é importante relembrar a função de cada grupo de bits: os *coset bits* serão utilizados como entrada para o código convolucional do decoder e os *index bits* serão utilizados apenas para selecionar, dentro do *coset* selecionado através da saída do código convolucional, o valor quantizado.

Prosseguindo com o decoder, o primeiro bit a ser utilizado como entrada para o código convolucional é o bit 0 (Figura 3.28(a)) resultando como saída do código convolucional os bits 00 e mudando o estado do código de 10 para 01 (Figura 3.28(b)). Utilizando o *index bit* correspondente (0), seleciona-se o valor -0.8750 dentro do *coset* $D0$ (Figura 3.28(b)).

Figura 3.23: Exclusão dos caminhos.



(a) Selação dos menores erros acumulados

(b) Exclusão dos maiores erros acumulados

Fonte: o autor.

O segundo bit do conjunto de *subset bits* a ser utilizado como entrada do código convolucional é 1 (Figura 3.29(a)) resultando, como saída, os bits 11 e movendo o estado do código convolucional de 01 para 11 (Figura 3.29(c)). Seleciona-se, portanto, o *coset* $D4$ pois os bits de saída são 11 e, utilizando o *index bit* correspondente (0) seleciona-se o elemento -0.1250 de $D4$ como mostra a Figura 3.29(c).

O terceiro e último bit do conjunto chamado *subset bits* a ser utilizado como entrada para o código convolucional é o bit 1 (Figura 3.30(a)) resultando, como saída, os bits 11 (Figura 3.30(c)). Estes bits da saída referenciam novamente o *coset* $D4$ e o elemento dentro de $D4$ que é selecionado pelo *index bit* correspondente (1) é 0.8750 (Figura 3.30(b)) resultando na sequência final Qtz_sig (abreviação em inglês de *sequência quantizada* mostrada a seguir:

$$Qtz\_seq = -0.8750, -0.1250, 0.8750. \tag{3.11}$$

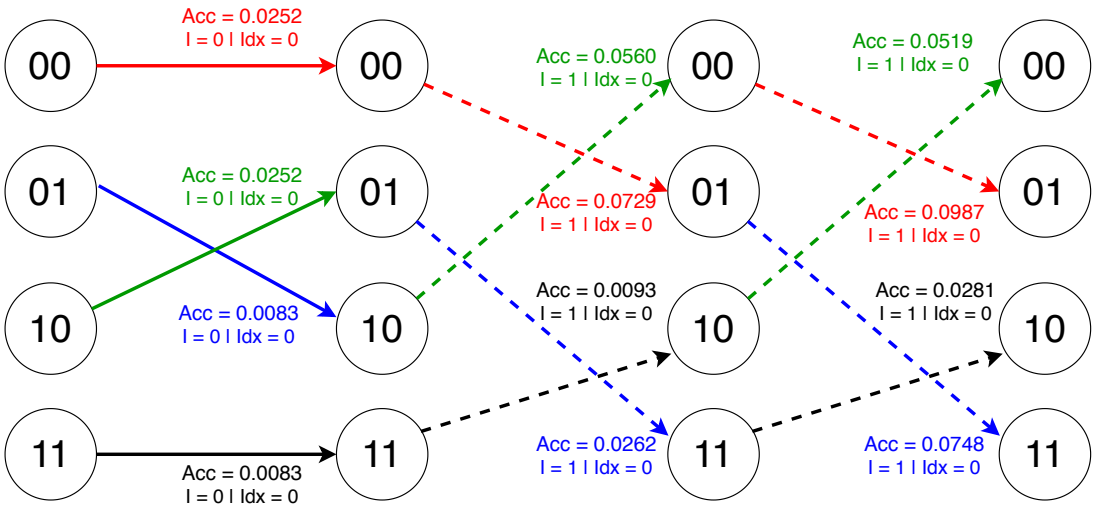
Nota-se que o uso dos dois bits iniciais que são utilizados para representar o estado inicial do código convolucional têm um peso cada vez menor na taxa de bits final a medida que o tamanho da sequência aumenta.

3.10 Custo computacional

Esta seção analisará o custo computacional do TCQ. Antes de realizar esta análise, é importante definir alguns parâmetros importantes como descrito a seguir:

- Será analisado apenas o custo necessário para quantizar uma amostra: esta medida é importante para normalizar em relação à quantização escalar, quantização vetorial e o LPC.
- Será avaliado a quantidade necessária de adições (ou subtrações), multiplicações e comparações.

Figura 3.24: Treliça finalizada.



Fonte: o autor.

- Não será contabilizado os custos da criação dos *cosets*.
- O número de *cosets* a ser utilizado nesta análise é quatro.

Com estes parâmetros definidos, neste momento será avaliado o custo total de adições, multiplicações e comparações necessários para quantizar uma amostra.

3.10.1 Primeira etapa: quantização escalar dos cosets

Como explicado na seção anterior, o primeiro passo a ser realizado quando uma amostra é emitida pela fonte é realizar a quantização utilizando os quatro *cosets* disponíveis. Cada *coset* contém exatamente $R-1$ elementos em que a variável R representa a taxa de bits. Como a quantização escalar é utilizada, esta operação de quantização pode ser realizada em $log_2(N)$ operações em que a variável N representa o número de *codewords* disponíveis. Como cada *coset* contém $2^{(R-1)}$ *codewords*, o número total de comparações necessárias da quantização escalar é:

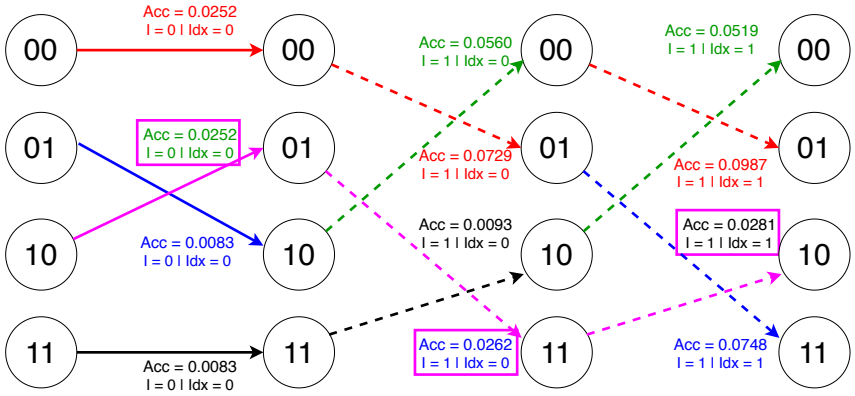
$$cmp = \log_2(N). \tag{3.12}$$

Substituindo N por $2^{(R-1)}$, temos que

$$\begin{aligned} cmp &= \log_2(2^{(R-1)}), \\ cmp &= (R-1). \end{aligned} \tag{3.13}$$

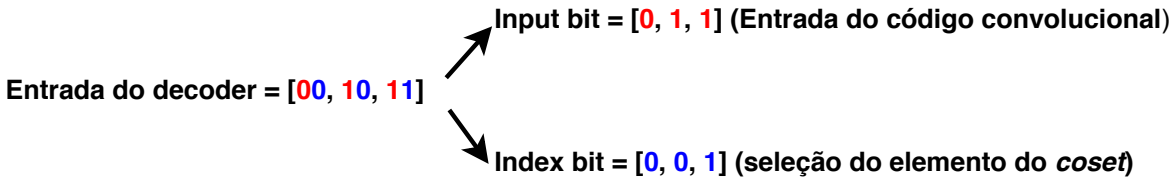
Portanto, na primeira etapa, $R-1$ comparações são necessárias por *coset*. Como são usados quatro *cosets*, portanto, $4(R-1)$ comparações são necessárias nesta etapa (Passo (I) da Figura 3.30).

Figura 3.25: Realizando o *Traceback*.



Fonte: o autor.

Figura 3.26: Separação dos *subset bits* e dos *index bit*.



Fonte: o autor.

3.10.2 Segunda etapa: adição do erro quadrático

Para cada quantização escalar realizada anteriormente, é necessário computar o erro quadrático entre o valor de entrada x e o valor quantizado x_q. Antes de prosseguir, é importante definir a Equação que calcula o erro. Para encontrar o erro quadrático d, usa-se a diferença quadrática, ou seja,

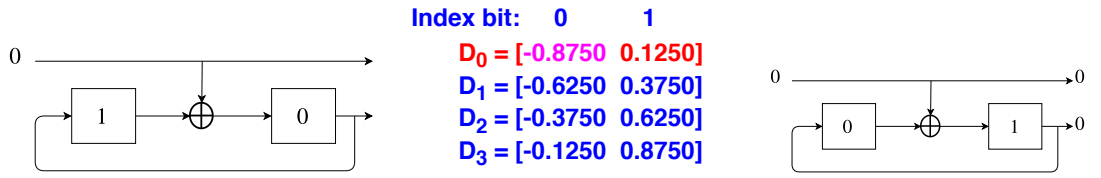
$$d = (x - x_q)^2. \tag{3.14}$$

Para realizar esta operação, nota-se que são necessários apenas uma adição (para simplificar a análise, tanto a subtração quanto a adição serão tratados como adição pois eles possuem o mesmo custo computacional) e uma mulplicação (Passo (II) da Figura 3.30).

3.10.3 Terceira Etapa: adição do erro quadrático no erro acumulado

Ao iniciar o percurso da treliça, nota-se sempre dois *path* saem de cada estado da treliça. É importante para o algoritmo de Viterbi adicionar o erro quadrático encontrado com o erro previamente acumulado. Como é observado na Figura 3.30, há sempre dois caminhos saindo de cada estado da treliça e, portanto, duas adições são necessárias (Passo (III) da Figura 3.30).

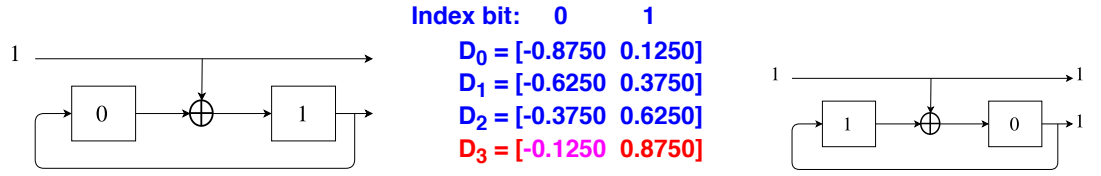
Figura 3.27: Estado do código convolucional antes e depois de processar o bit 0.



(a) Sequência a ser quantizada. (b) Codebook. (c) Estado final do código.

Fonte: o autor.

Figura 3.28: Estado do código convolucional antes e depois de processar o bit 0.



(a) Sequência a ser quantizada. (b) Codebook. (c) Estado final do código.

Fonte: o autor.

3.10.4 Quarta Etapa: eliminação dos caminhos desnecessários

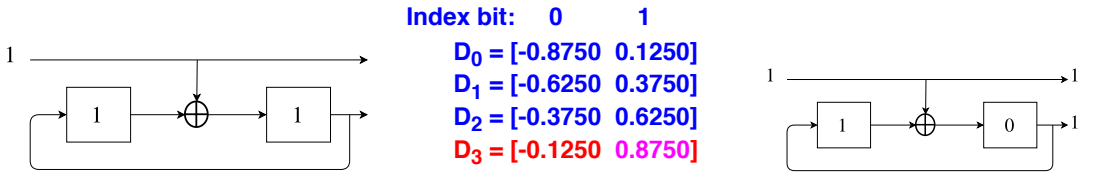
Como para cada estado da treliça há sempre dois caminhos chegando, é importante sempre eliminar o caminho mais custoso. Esta operação pode ser realizada com apenas uma compação para cada estado da treliça. Como, neste exemplo, a treliça possue quatro estados, então quatro comparações são necessáras (Passo (IV) da Figura 3.30).

3.10.5 Custo computacional: Decoder

Como ilustrado em [28], o custo computacional do decoder é bem menor do que o encoder e, portanto, nas análises computacionais do TCQ, utiliza-se somente o custo computacional do encoder.

Portanto, o custo computacional total é a soma dos custos das quatro etapas descritas anteriormente. Este custo depende linearmente.

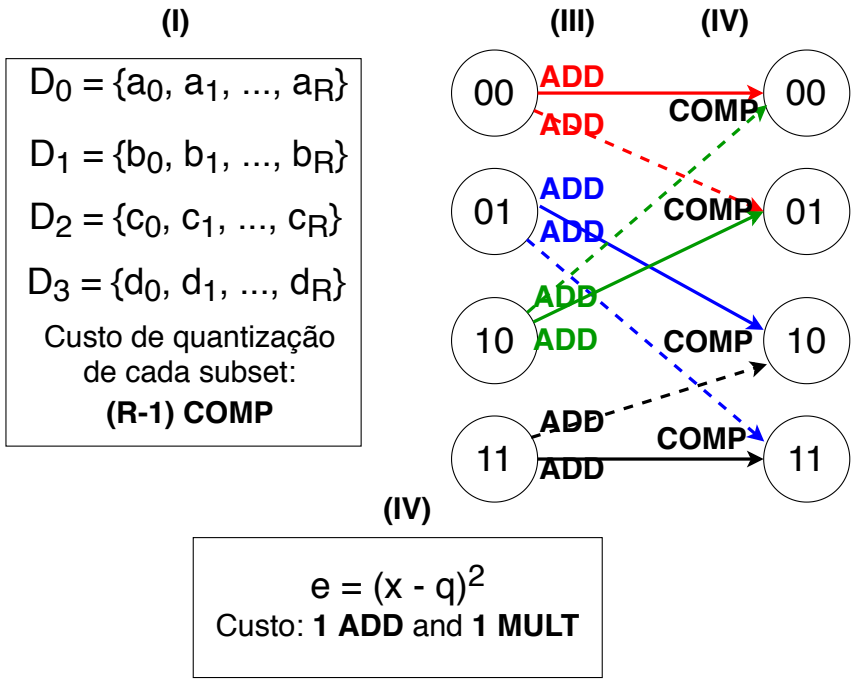
Figura 3.29: Estado do código convolucional antes e depois de processar o bit 0.



(a) Sequência a ser quantizada. (b) Codebook. (c) Estado final do código.

Fonte: o autor.

Figura 3.30: Análise do custo computacional do TCQ.



Fonte: o autor.

Capítulo 4

EXPERIMENTOS E RESULTADOS

Neste capítulo será abordado os resultados obtidos neste trabalho. Primeiramente, os sinais utilizados serão descrito bem como todo os parâmetros necessários para a realização dos experimentos.

4.1 Descrição dos sinais utilizados

A Tabela 4.1 abaixo ilustra características dos sinais utilizados neste experimento.

Para os próximos experimentos, o sinal C1 é utilizado como treino enquanto o sinal C2 é utilizado como teste. Para projetar os codebooks, o algoritmo de Lloyd é utilizado. Os outros sinais serão utilizados no experimento descrito na seção 4.6.

Na primeira etapa dos experimentos, avaliou-se a performance do sistema descrito na Figura 4.1 primeiramente utilizando o TCQ, quantização escalar (SQ) e quantização vetorial (VQ) na etapa de quantização (Bloco do encoder e decoder da Figura 4.1.

4.2 Resultados utilizando TCQ

Para os resultados utilizando TCQ como quantizador, avaliou-se a performance para diferentes treliças. Todas as treliças utilizadas nestes experimentos foram propostas por Ungerboeck [26] e [27] e possuem 4, 8, 16, 32, 64, 128 e 256 estados. Para fins de simplificação, a seguinte nomenclatura será utilizada:

1. **TCQ-4:** TCQ utilizando a treliça de 4 estados.
2. **TCQ-8:** TCQ utilizando a treliça de 8 estados.
3. **TCQ-16:** TCQ utilizando a treliça de 16 estados.
4. **TCQ-32:** TCQ utilizando a treliça de 32 estados.
5. **TCQ-64:** TCQ utilizando a treliça de 64 estados.
6. **TCQ-128:** TCQ utilizando a treliça de 128 estados.

Tabela 4.1: Descrição dos sinais utilizados.

ID	Modulação	Resource Blocks Ativos	Modo Duplex	Prefixo Cíclico	Largura de Banda (MHz)
C1	64 QAM	100	FDD	Normal	20
C2	16 QAM	100	FDD	Normal	20
C3	4 QAM	100	FDD	Normal	20
C4	16 QAM	10	FDD	Normal	20

Fonte: o autor.

7. **TCQ-256:** TCQ utilizando a treliça de 256 estados.

A Figura 4.2 ilustra os resultados obtidos em termos de *Error Vector Magnitude* (EVM) []. Como é possível observar na Figura 4.2, aumentar o número de estados da treliça não resulta em uma melhora considerável do sistema. Portanto, para os próximos experimentos utilizando TCQ como quantizador, será utilizado o TCQ-4 pois possui o menor custo computacional dentre todos os blocos.

O próximo passo consiste em avaliar o sistema utilzando TCQ-4 para diferentes valores do tamanho da linha do *Block Scaling* descrito na Figura 4.1. O tamanho da linha influencia na compressão do sistema, pois como cada linha é normalizada pelo seu valor máximo, aumentar o tamanho da linha aumenta a possibilidade de um valor que está localizado nos extremos da distribuição de probabilidade do sinal [21].

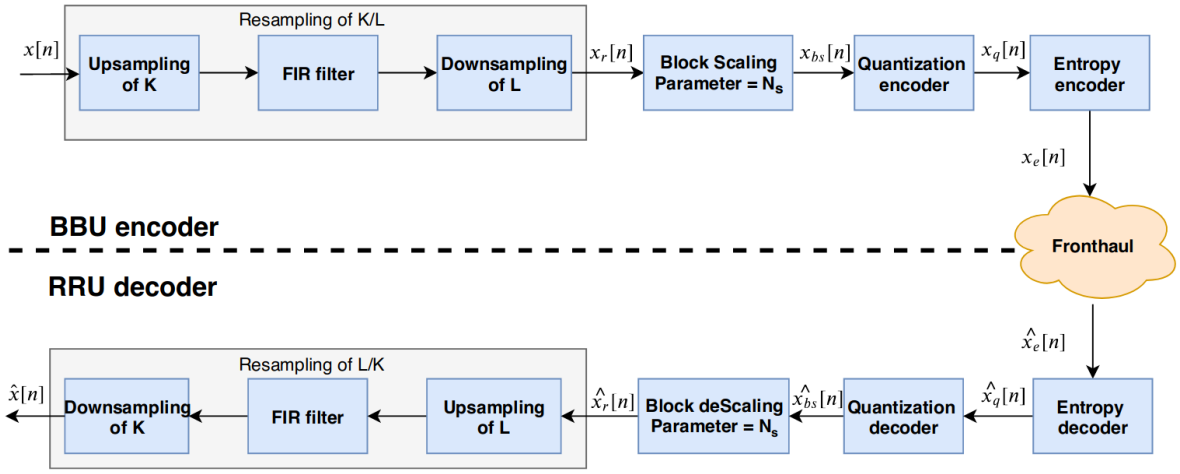
A Figura 4.3 retrata que aumentar o tamanho da linha do *Block Scaling* resulta em uma melhora do sistema. Este resultado pode ser explicado pois o TCQ foi projetado para quantizar sequências e aumentar o tamanho da sequência melhora a performance do quantizador. Entretanto, como é possível observar na Figura 4.3, aumentar indefinidamente o tamanho da sequência não melhora a performance do sistema da mesma forma pois, a partir de uma sequência de 256 elementos, obsevar-se uma saturação na performance do sistema. Portanto, para os próximos experimentos utilizando o TCQ como quantizador será utilizado o TCQ-4 quantizando uma sequência de 256 elementos.

4.3 Resultados utilizando quantizador escalar

Os próximos resultados ilustram a performance do quantizador escalar no sistema da Figura 4.1. Este quantizador é o mais simples e menos custoso dos avaliados até nestes experimentos.

A Figura 4.4 mostra a performance do quantizador para diferentes valores da sequência. Como é possível observar na Figura 4.4, aumentar indefinidamente o tamanho da sequência não resulta em uma melhora do sistema da mesma forma. Observa-se uma saturação da performance dos sistemas para sequências maiores de 256 elementos. Portanto, para os próximos

Figura 4.1: Diagrama de blocos do sistema utilizado nos experimentos.



Fonte: o autor.

experimentos utilizando o quantizador escalar, o tamanho da sequência será fixado em 256 elementos.

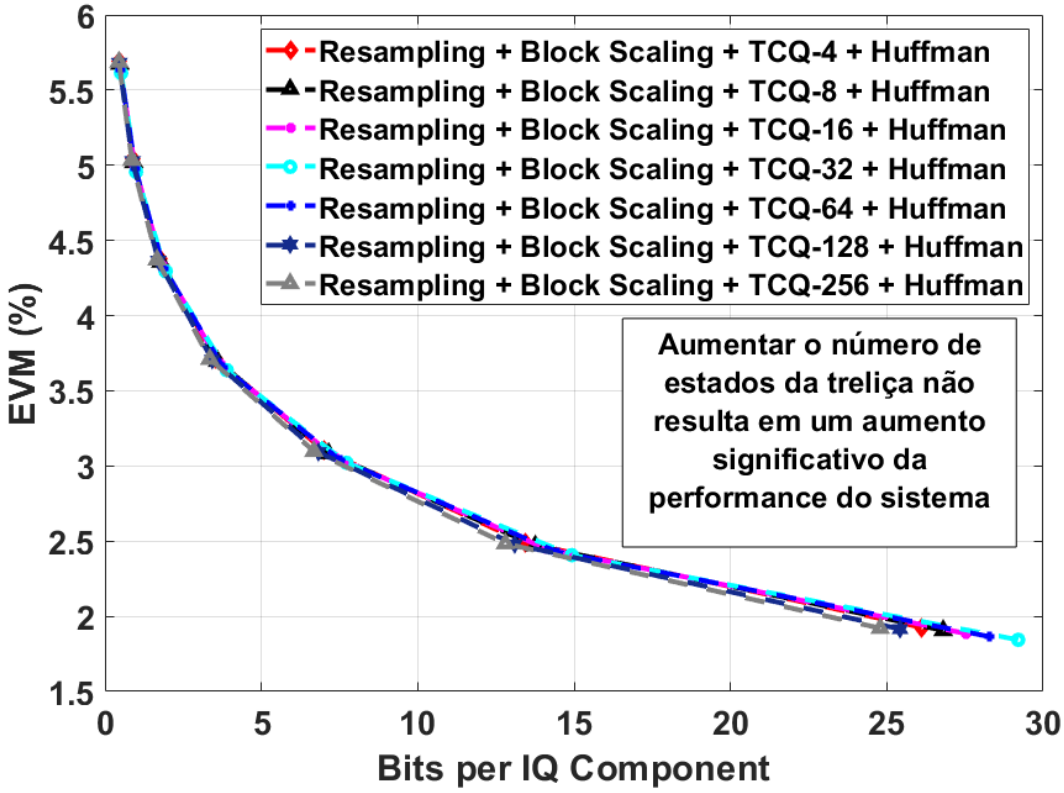
4.4 Resultados utilizando quantizador vetorial

O quantizador vetorial descrito neste experimento é o quantizador de duas dimensões. Ele será chamado de VQ-2 (o número 2 faz referência a dimensão do quantizador). É possível notar observando a Figura 4.5 que a performance do sistema apresenta a mesma saturação para sequências superior a 256 elementos.

4.5 Comparativo entre os quantizadores TCQ-4, SQ e VQ-2

Finalizando a primeira etapa dos experimentos, a Figura 4.7 apresenta um comparativo da performance do sistema para os três quantizadores utilizados: TCQ-4, SQ e VQ-2. Como é possível observar na Figura 4.6, o TCQ apresenta a melhor performance. Outro fator interessante está no fato do quantizador vetorial possuir o maior custo computacional entre os três. Como ilustração, a Figura 4.7 apresenta um comparativo entre os quantizadores VQ-2 e TCQ-4. É possível obsevar que, para taxas maiores, utilizar VQ-2 resulta em um grande custo computacional tornando um fator determinante contra este quantizador. Ao contrário do VQ-2, o TCQ-4 apresenta um crescimento linear e, para taxas maiores, é preferível utilizar TCQ-4 ao invés da VQ-2 mesmo que a EVM do TCQ-4 seja maior.

Figura 4.2: Performance do sistema utilizando TCQ para diversas treliças.



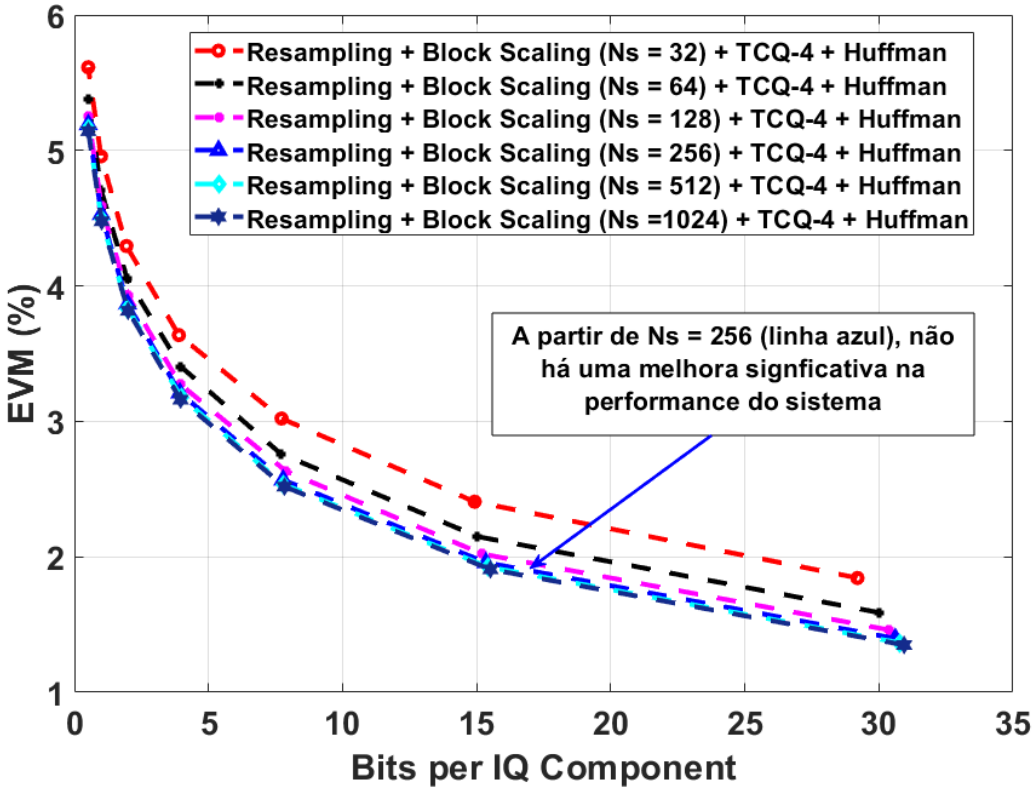
Fonte: o autor.

4.6 Performance para vários sinais

A seguir, ilustram-se os resultados obtidos quando diferentes sinais de testes foram utilizados. Os sinais C2 (16 QAM e 100 *resource blocks* ativos); C3 (4 QAM e 100 *resource blocks* ativos) e C4 (64 QAM e 10 *resource blocks* ativos) foram utilizados como teste enquanto o sinal C1 ((64 QAM e 100 *resource blocks* ativos). Para este experimento, utilizou-se o quantizador de 6 bits.

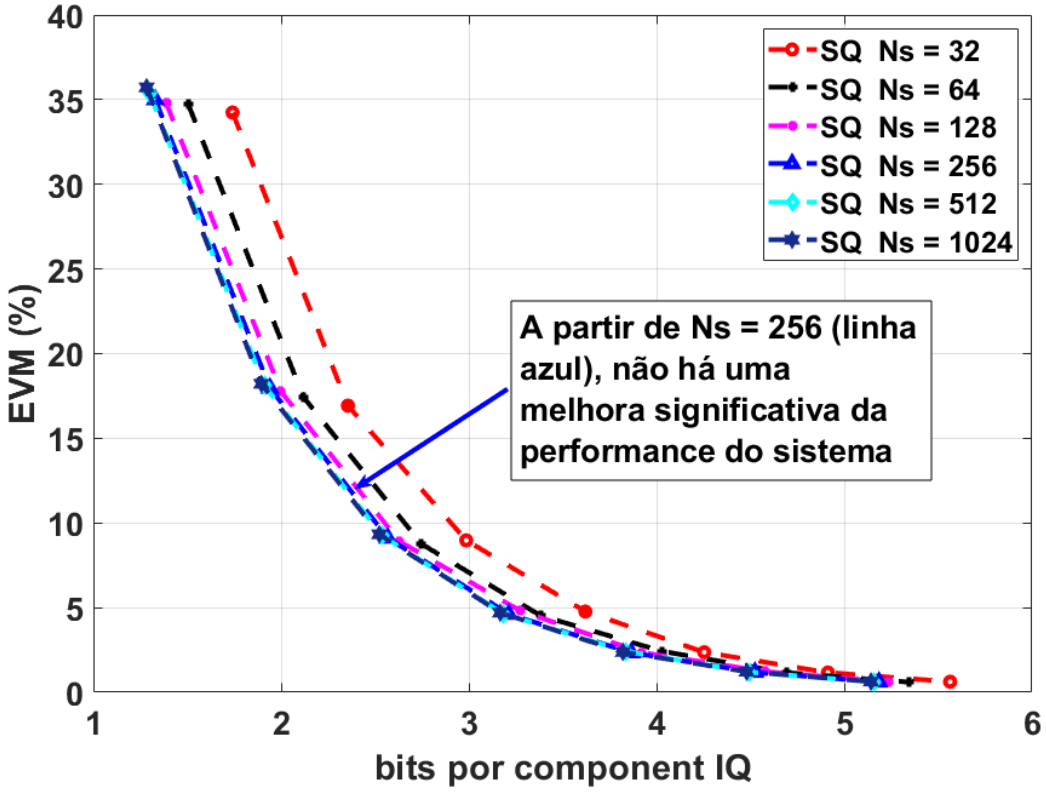
A Tabela 4.2 mostra a porcentagem da EVM média obtida e a taxa em bits por component IQ do sistema obtidos. Como é possível observar, todos os sistemas de compressão obtiveram um resultado similar para os sinais de teste C2 e C3 enquanto, para o sinal C4, houve um aumento da EVM. Entretanto, para o sinal C4 houve uma compressão maior. Isto se deve ao fato do sinal C4 ter menos resource blocks ativos e, portanto, menos informação é transmitida necessitando assim de menos bits.

Figura 4.3: Performance do sistema utilizando TCQ para diversas tamanhos da sesquência.



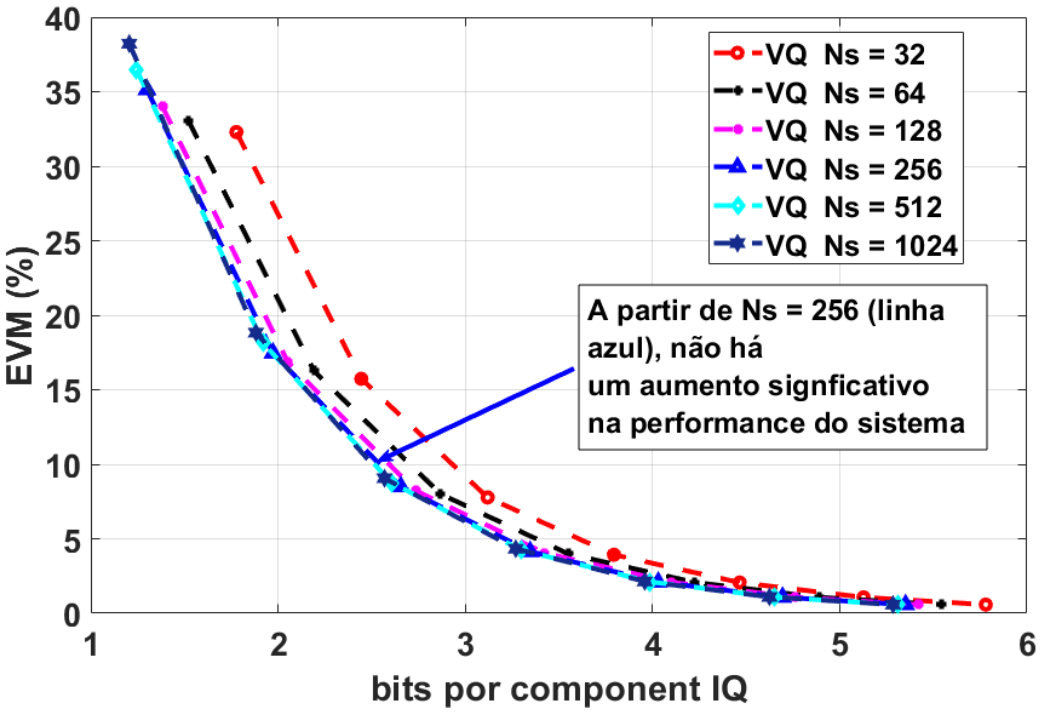
Fonte: o autor.

Figura 4.4: Performance do sistema utilizando quantizador escalar para diversas treliças.



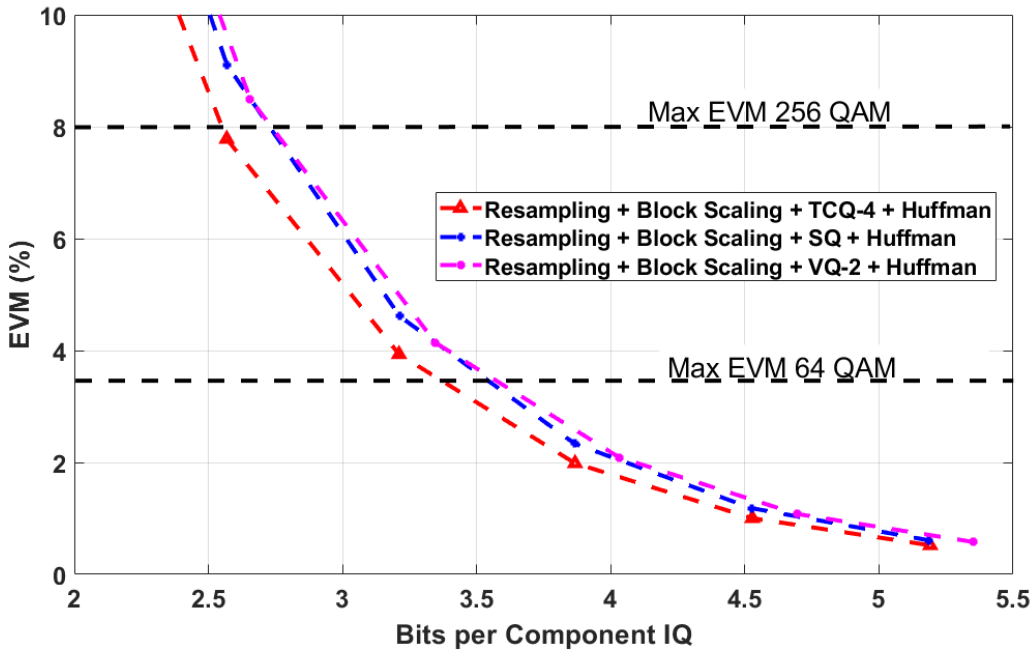
Fonte: o autor.

Figura 4.5: Performance do sistema utilizando quantizador vetorial (VQ) para diversos tamanho s da sequência.



Fonte: o autor.

Figura 4.6: Comparativos entre os quantizadores SQ.



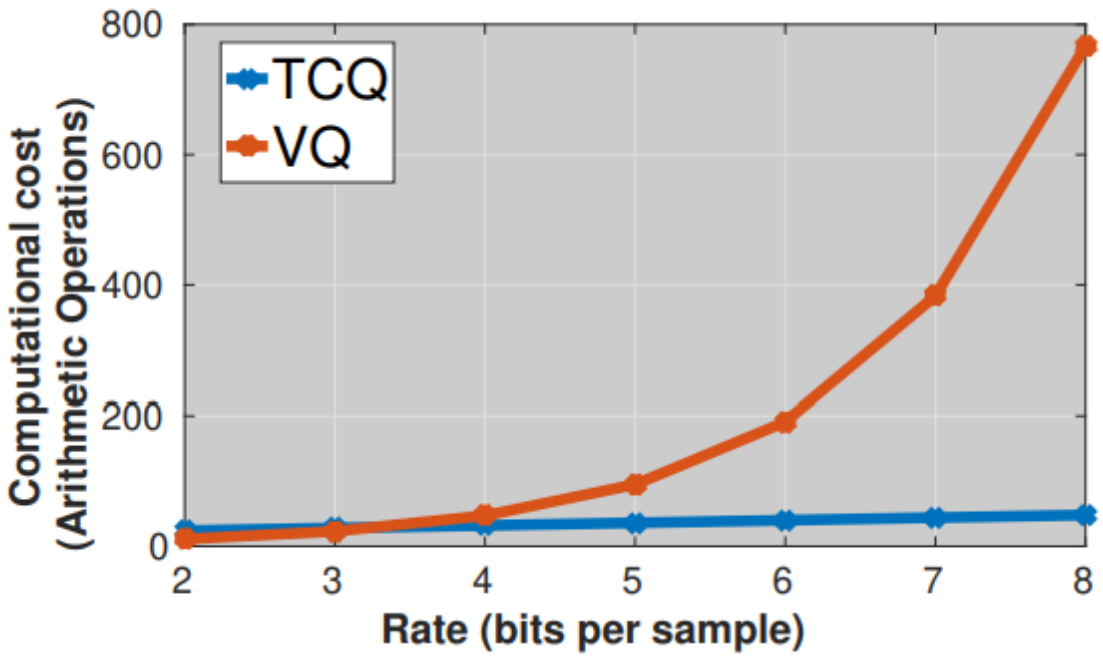
Fonte: o autor.

Tabela 4.2: Resultados para diferentes sinais.

Train ID: C1	**Test ID**		
	C2	**C3**	**C4**
SQ EVM (%)	2.33	2.33	3.55
SQ rate	3.86	3.86	3.91
TCQ-4 EVM (%)	1.98	1.98	3.00
TCQ-4 rate	3.86	3.86	3.91
VQ-2 EVM (%)	2.09	2.09	3.65
VQ-2 rate	4.03	4.03	3.98
LPC EVM (%)	1.65	1.65	1.65
LPC rate	4.39	4.39	3.85

Fonte: o autor.

Figura 4.7: Custo Computational do TCQ-4 e da VQ-2.



Fonte: o autor.

Capítulo 5

CONCLUSÃO

Neste trabalho apresentou-se diferentes sistemas de quantização para quantização de sinal no fronthaul. Como visto, é importante comprimir os sinais que são transportados para que os requerimentos das redes 4G e 5G sejam atingidas.

Este trabalho iniciou explicando a evolução das redes móveis, o que levou a atual arquitetura que é utilizada neste trabalho para compressão.

Diferentes técnicas foram propostas para realizar esta compressão. Dentre estas técnicas, destaca-se a proposta em [19] que realiza a predição linear das amostras de um símbolo OFDM e quantiza o erro de predição. Em [20] e [21], o sistema proposto para quantizar é baseado em quatro blocos: 2/3 resampling, Block Scaling, Quantizador e codificação Huffman. A diferença entre os trabalhos é que o primeiro utiliza quantizador escalar enquanto o segundo utiliza quantização vetorial.

Neste trabalho, realizou-se a simulação dos sistemas de compressão baseada na técnica do resampling. Como fator inovador, um novo sistema de compressão foi proposto. Este sistema segue a mesma estrutura dos trabalhos [20] e [21]. A diferença é que este trabalho utiliza TCQ como quantizador.

Como experimentos, avaliou-se o sistema que utilizar um resampling de $2/3$ seguido de Block Scaling; quantização e codficação Huffman. Três quantizadores foram propostos para o bloco do encoder e decoder: TCQ, SQ e VQ-2.

Para o TCQ, avaliou-se a performance do sistema para treliças com 4, 8, 16, 32, 64, 128 e 256 estados. Os experimentos mostraram que aumentar o tamanho da treliça não resulta em uma melhora significativa do sistema. Portanto, para os experimentos seguintes, optou-se por utilizar o TCQ com a treliça de 4 estados pois possui o menor custo computacional dentre as opções.

Com o tamanho da treliça fixada, optou-se por verificar como o sistema se comporta para diferentes tamanhos das sequências (32, 64, 128, 256, 512 e 1024). Como foi mostrado, a partir da sequência de 256 elementos, observou-se uma saturação na performance para todos os quantizadores avaliados neste sistema.

Estes resultados conduziram a comparação dos três quantizadores (TCQ-4, SQ e VQ-2). Os resultados mostraram que utilizar o quantizador TCQ-4 oferece a melhor performance e, como mostrado no capítulo 3, o TCQ-4 possui um custo computacional que cresce linearmente e, portanto, também oferece um bom custo computacional (Ao contrário da quantização vetorial cujo custo computacional cresce exponencialmente).

5.1 Trabalhos Futuros

Como trabalhos futuros, as seguintes observações serão feitas:

- **Utilizar *Predictive TCQ*:** esta técnica foi proposta em [28] e é uma versão do TCQ na qual utiliza a combinação de predição linear com treliças propostas por Ungerboeck. O *Predictive TCQ* será utilizado para comprimir os símbolos OFDM pois há a presença de correlação entre as amostras para cada símbolo.
- **Utilizar outras treliças além das propostas:** utilizar diferentes treliças das propostas por Ungerboeck podem resolver o problema da saturação como visto no capítulo anterior, resultando em uma melhora na performance do sistema.
- **Utilizar diferentes codebooks:** a saturação observada nos experimentos pode ter relação com o codebook que foi projetado para cada um dos quantizadores. Portanto, diferentes codebooks podem evitar a saturação observada ao aumentar o tamanho da sequência observada nas Figuras 4.3, 4.4 e 4.5
- **Utilizar Quantizador Vetorial de 3 dimensões:** apesar do aumento do custo computacional, utilizar um quantizador vetorial de três dimensões pode ser vantajoso para determinadas taxas.

5.2 Publicações

Este trabalho resultou em artigo em inglês intitulado *"A Fronthaul Signal Compression Method Based on Trellis Coded Quantization"*. Este artigo foi publicado na conferência internacional do IEEE chamada *"Latin-American Conference on Communications"* (LATINCOM).

Além deste trabalho principal, as seguintes publicações foram realizadas durante o período do Mestrado:

1. **Flávio Brito**, Andrey Silva, Leonardo Ramalho and Aldebaro Klautau. *Compression of MIMO FDD Channel State Information Using a Trellis-Based Technique*. Conferência Nacional em Comunicações, Redes e Segurança da Informação (ENCOM 2019)
2. **Flávio Brito**, Andrey Silva, Kaio Forte, Carnot Braun, Diego Dantas, Silvia Lins, Neiva Linder, and Aldebaro Klautau. *Testbed Development and Evaluation of Drone-based Real-time Object Detection using Deep Learning.* Latin American Student Workshop on Data Communication Networks (LANCOMM 2019)
3. Diego Dantas, Carnot Braun, Kaio Forte, **Flavio Brito**, Andrey Silva, Silvia Lins, Neiva Linder, and Aldebaro Klautau. *Testbed Development of Human Pose Estimation using Deep Learning for Unmanned Aerial Vehicles*. Congresso Brasileiro de Inteligência Computacinonal (CBIC 2019)
4. Ingrid Nascimento, **Flavio Mendes**, Marcus Dias, Andrey Silva, and Aldebaro Klautau. *Deep Learning in RAT and Modulation Classification with a New Radio Signals Dataset*. Brazilian Symposium on Telecommunications and Signal Processing (SBRT 2018)

Referências Bibliográficas

[1] C. Cox, *An introduction to LTE: LTE, LTE-advanced, SAE and 4G mobile communications*. John Wiley & Sons, 2012.

[2] S. Sesia, I. Toufik, and M. Baker, *LTE-the UMTS long term evolution: from theory to practice*. John Wiley & Sons, 2011.

[3] E. Traffic, "Trafic and market data report," 2013.

[4] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang, "What will 5g be?" *IEEE Journal on selected areas in communications*, vol. 32, no. 6, pp. 1065–1082, 2014.

[5] E. Traffic, "Ericsson Mobility Report Q2 Update 2012," 2013.

[6] "Ericsson Mobility Report Q2 Update2019," p. 4, 2019.

[7] 3GPP, "Requirements for evolved utra (e-utra) and evolved utran (e-utran)," 2013.

[8] ——, "Requirements for further advancements for evolved universal terrestrial," 2013.

[9] ——, "3gpp – releases (accessed 12 december, 2011)." 2013.

[10] ——, "Evolved universal terrestrial radio access network (e-utran: Architecture description," 2011.

[11] ——, "3rd generation partnership project (2011) 3gpp." 2013.

[12] ——, "Requirements for evolved utra (e-utra) and evolved utran (e-utran), release 8," 2013.

[13] I.-H. Kim, "Why do we need antenna-integrated RRH (Remote Radio Antenna, RRA)?" NETMANIAS TECH-BLOG, mar 2015.

[14] I. Chih-Lin, J. Huang, R. Duan, C. Cui, J. X. Jiang, and L. Li, "Recent progress on c-ran centralization and cloudification," *IEEE Access*, vol. 2, pp. 1030–1039, 2014.

[15] B. Sklar and F. J. Harris, *Digital communications: fundamentals and applications*. Prentice-hall Englewood Cliffs, NJ, 1988, vol. 2001.

[16] A. Gersho and R. M. Gray, *Vector quantization and signal compression*. Springer Science & Business Media, 2012, vol. 159.

[17] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[18] J. Max, "Quantizing for minimum distortion," *IRE Transactions on Information Theory*, vol. 6, no. 1, pp. 7–12, 1960.

[19] L. Ramalho, M. N. Fonseca, A. Klautau, C. Lu, M. Berg, E. Trojer, and S. Höst, "An lpc-based fronthaul compression scheme," *IEEE Communications Letters*, vol. 21, no. 2, pp. 318–321, 2016.

[20] B. Guo, W. Cao, A. Tao, and D. Samardzija, "Lte/lte-a signal compression on the cpri interface," *Bell Labs Technical Journal*, vol. 18, no. 2, pp. 117–133, 2013.

[21] H. Si, B. L. Ng, M. S. Rahman, and J. Zhang, "A novel and efficient vector quantization based cpri compression algorithm," *IEEE Transactions on vehicular technology*, vol. 66, no. 8, pp. 7061–7071, 2017.

[22] A. V. Oppenheim, *Discrete-time signal processing*. Pearson Education India, 1999.

[23] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[24] G. Ungerboeck, "On improving data-link performance by increasing channel alphabet and introducing sequence coding," in *Int. Symp. Inform. Theory*, 1976.

[25] ——, "Channel coding with multilevel/phase signals," *IEEE transactions on Information Theory*, vol. 28, no. 1, pp. 55–67, 1982.

[26] ——, "Trellis-coded modulation with redundant signal sets part i: Introduction," *IEEE Communications magazine*, vol. 25, no. 2, pp. 5–11, 1987.

[27] ——, "Trellis-coded modulation with redundant signal sets part ii: State of the art," *IEEE Communications magazine*, vol. 25, no. 2, pp. 12–21, 1987.

[28] M. W. Marcellin and T. R. Fischer, "Trellis coded quantization of memoryless and gauss-markov sources," *IEEE transactions on communications*, vol. 38, no. 1, pp. 82–93, 1990.