UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

EDUARDO SACHSER

Aplicação de técnicas de compressed sensing e aprendizado de dicionários para codificação de vídeos em tempo real

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Manuel Menezes de Oliveira Neto

Porto Alegre
2018

RESUMO

Milhões de vídeos circulam pela *internet* diariamente. Para que isso seja possível, é necessário que existam e sejam desenvolvidos métodos de codificação capazes de comprimir vídeos, aliando compressibilidade a qualidade na representação. Este trabalho propõe uma metodologia para codificação e decodificação de vídeos em tempo real, utilizando técnicas de *compressed sensing* e aprendizado de dicionários, além de quantização e compressão utilizando codificação de Huffman. Do ponto de vista teórico, o trabalho apresenta os conceitos fundamentais de representação de sinais, além de aprofundar os assuntos de *compressed sensing* e aprendizado de dicionários, mostrando alguns resultados alcançados e técnicas aplicadas a cada um dos assuntos. Também são abrangidos outros trabalhos que têm relação com a proposta, sobre os quais são discutidas as ideias principais, bem como resultados alcançados, e de que maneira se relacionam com o trabalho proposto. Nos experimentos realizados, são discutidas as parametrizações possíveis do codec proposto, apresentando resultados experimentais alcançados e recomendações relativas ao uso do codec. O trabalho ainda compara o amplamente utilizado padrão de codificação e decodificação de vídeos H.264 com a proposta, através das métricas de bitrate e PSNR. Os resultados obtidos são inferiores ao padrão H.264 quando comparado o PSNR alcançado por ambos para um mesmo bitrate. O trabalho permite a compreensão dos assuntos de *compressed sensing* e aprendizado de dicionários, aplicados ao problema de codificação de vídeos.

Palavras-chave: Compressed sensing. compressive sampling. aprendizado de dicionários. tempo real. video. UFRGS.

An application of compressed sensing and dictionary learning techniques for real-time video coding and decoding

ABSTRACT

Millions of videos run through the internet every day. To make this possible, coding methods must be developed, this ones capable of compress video data, combining compressibility and quality of representation. This work proposes a method for real-time video coding and decoding, using compressed sensing and dictionary learning techniques, besides of quantization and compression based on Huffman coding. From a theoretical point of view, this work introduces the fundamental concepts of signal representation, in addition to deepening the subjects of compressed sensing and dictionary learning, showing some achieved results and techniques applied to each of the subjects. Also are covered works related to the proposal, on which the main ideas are discussed, as well the achieved results, and how they relate to the proposed work. In the experimentation are discussed the possible parameterizations of the proposed codec, showing achieved experimental results and recomendations related to the manner of using the codec. The work also compares the widely used video coding and decoding standard H.264 with the proposal, through bitrate and PSNR metrics. The obtained results are lower than the H.264 standard when compared the achieved PSNR by the both with same bitrates. This work allows to comprehend the subjects of compressed sensing and dictionary learning, applying to video coding.

Keywords: compressed sensing. compressive sampling. dictionary learning. real-time. UFRGS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CoDec	Coding and Decoding
CoSaMP	Compressive Sampling Matching Pursuit
CS	Compressed Sensing
DCT	Discrete Cosine Transform
FPS	Frames por segundo
GPU	Graphical Processing Unit
ODL	Online Dictionary Learning
OMP	Orthogonal Matching Pursuit
PSNR	Peak signal-noise ratio
RIP	Restricted Isometric Property

SUMÁRIO

1 INTRODUÇÃO

Segundo a revista Forbes (MCCUE, 2018), mais de 500 milhões de horas de vídeos são assistidos diariamente no Youtube. Ainda segundo a revista, até 2021, 17.000 horas de conteúdo de vídeo passarão pela *internet*, por segundo. Isso somente é possível graças a eficiência dos padrões de compressão e transmissão de vídeo e áudio criados ao longo dos anos, e ainda sendo estudados e melhorados. Diversos padrões foram criados, e com o avanço tecnológico, mais tecnologias são possíveis de serem aplicadas a novos padrões, permitindo que mais informação trafegue na rede com menor quantidade de dados sendo necessários.

Dentro desse contexto, este trabalho explora o uso de *compressed sensing* e aprendizado de dicionários (*dictionary learning*), para codificação de vídeos em tempo real. O trabalho também avalia o desempenho da técnica implementada, comparando-o com o do codificador H.264.

1.1 Objetivos

Este trabalho tem os seguintes objetivos:

- Apresentar uma proposta de codificação de decodificação de vídeos em tempo real baseada em técnicas de *compressed sensing* e aprendizado de dicionários;
- Realizar medidas comparativas entre as diversas possibilidades de parametrização do objeto do trabalho proposto;
- Realizar medidas comparativas com outros padrões conhecidos de codecs de vídeo;
- Aprofundar o conhecimento da área de *compressed sensing* e aprendizado de dicionários através aplicação de técnicas relacionadas a tais assuntos.

1.2 Notações Utilizadas

Algumas notações serão utilizadas ao longo de todo o trabalho. Para representar matrizes, serão utilizadas letras gregas ou romanas, maiúsculas e em negrito, por exemplo $\boldsymbol{X}$. As colunas de $\boldsymbol{X}$ serão representadas pela letra correspondente minúscula em negrito, com o indexador da coluna, $\boldsymbol{x_i}$, também em negrito. Outra forma possível de representar partes de uma matriz é $\boldsymbol{X}_{:,i}$, que representa todas as linhas da coluna i, e $\boldsymbol{X}_{i,:}$, que re-

presenta todas as colunas da linha i. Um valor qualquer da matriz é descrito como $\boldsymbol{X_{i,j}}$. Letras minúsculas em negrito representarão vetores. A indexação de vetores acontecerá no formato x_i, não utilizando negrito.

Alguns algoritmos calculam variáveis relacionadas a uma determinada iteração t. Nesses algoritmos, seguindo as notações anteriormente apresentadas, $\boldsymbol{X_{(t)}}$ representa a matriz $\boldsymbol{X}$ na t-ésima iteração. O mesmo acontece com vetores, por exemplo, $\boldsymbol{x_{(t)}}$.

Além dessas definições, é utilizado o conceito de produto interno de dois vetores $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$, conforme a seguinte notação:

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{m} x_i y_i.$$

Por fim, vale lembrar a definição da norma ℓ_p para um vetor $\boldsymbol{x} \in \mathbb{R}^m$:

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{m} |x_i|^p \right)^{\frac{1}{p}},$$

podendo escrever, portanto, a norma ℓ_2, por exemplo, como:

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{m} x_i^2},$$

e a norma ℓ_1:

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{m} |x_i|.$$

As normas acima apresentadas, bem como as notações, serão amplamente utilizadas em todo o texto. Outras definições serão agregadas ao longo do texto, somando-se as atuais, que foram apresentadas por serem aplicadas em toda a extensão do trabalho.

1.3 Estrutura do Trabalho

O Capítulo 2 apresenta os conceitos fundamentais que serviram de base para a elaboração da proposta do trabalho, abordando assuntos relativos a processamento de sinais, *compressed sensing* e aprendizado de dicionários. No Capítulo 3 discutem-se alguns trabalhos que inspiraram a proposta ou que apresentam diferentes óticas para o mesmo problema. A seguir, será detalhada a proposta. Baseando-se na mesma, serão apresenta-

dos alguns resultados de experimentos aplicados ao objeto do trabalho, encerrando com conclusões tiradas com base na proposta, nos trabalhos relacionados, e nos experimentos executadas.

2 CONCEITOS FUNDAMENTAIS

2.1 Introdução a Sinais e Sistemas

O trabalho apresenta uma metodologia para compressão de vídeos. Porém, vídeos são, particularmente, uma sequência de imagens. Já imagens, por sua vez, são um exemplo de sinais bidimensionais. Vídeos, portanto, sinais tridimensionais. Fica claro, a partir do posto acima, que há a necessidade de observar-se o que é um sinal e alguns resultados importantes, primeiramente, para em seguida aprimorarem-se os estudos acerca de outros conceitos.

"Um sinal é formalmente definido como uma função de uma ou mais variáveis, a qual veicula informações sobre a natureza de um fenômeno físico" (HAYKIN; VEEN, 2001). Também segundo Haykin and Veen (2001), os sinais são chamados unidimensionais quando dependem de apenas uma variável, e multidimensionais quando dependem de mais que uma. Como exemplo de sinais unidimensionais existem os sinais de som, dependentes apenas da variável tempo. Já exemplos de sinais multidimensionais foram citadas anteriormente, imagens e vídeos.

Além disso, deve-se definir o conceito de "sistema". Segundo Haykin and Veen (2001), "Um sistema é definido como uma entidade que manipula um ou mais sinais para realizar uma função, produzindo novos sinais". Um exemplo simples poderia ser o controle de volume de um aparelho de som, que gera um novo sinal proporcional ao sinal de entrada. Podemos considerar sistemas bem mais complexos também, como um reconhecedor de faces, que precisa tomar como sinais as faces já conhecidas e a que se deseja reconhecer, e fornece como saída o nome ou as informações da face reconhecida.

Sinais podem ser contínuos - ou analógicos - e digitais. Na verdade, em geral, sinais da natureza são contínuos, mas são digitalizados para que seja possível armazená-los, enviá-los e transmiti-los por meios digitais. Dessa forma, Haykin and Veen (2001) apresentam os passos geralmente executados para digitalizar sinais analógicos:

1. *Amostragem:* O sinal analógico é amostrado com determinada taxa de amostragem;
2. *Quantização:* Cada uma das amostras é transformada para um valor digital em um intervalo definido;
3. *Codificação:* Os valores quantizados são salvos de alguma forma;
4. *Redundância:* Podem ser geradas informações de redundância para armazenamento do sinal.

De uma forma geral, a aquisição ou até mesmo a transmissão de um sinal podem conter ruído. Dessa forma, pode-se imaginar que um sinal amostrado é composto pelo próprio sinal somado ao ruído, $\boldsymbol{f} = \boldsymbol{y} + \boldsymbol{e}$, no qual $\boldsymbol{f}$ é a amostragem, $\boldsymbol{y}$ é o sinal real, e $\boldsymbol{e}$ é o erro.

Uma definição importante para o prosseguimento do trabalho é a de energia E de um sinal $\boldsymbol{f} \in \mathbb{R}^m$, como segue:

$$E = \sum_{i=1}^{m} f_i^2 = \|\boldsymbol{f}\|_2^2, \tag{2.1}$$

onde f_i representa a i-ésima componente do vetor $\boldsymbol{f}$.

Por fim, um resultado clássico sobre amostragem de sinais no domínio tempo, o teorema de Nyquist.

Teorema 1. *(NYQUIST, 1928). Um sinal contínuo pode ser apropriadamente amostrado somente se a máxima frequência f_{max} contida no sinal for menor que metade da taxa de amostragem f_s:*

$$f_{max} < \frac{f_s}{2}.$$

—

2.2 Compressed Sensing

Compressed Sensing (CS) ou *Compressive Sampling* é um campo de estudo de sinais cujos primeiros trabalhos surgiram a partir de 2006, a partir da análise de amostragens executadas em imagens de ressonância magnética. Essas amostragens eram de muito menor dimensionalidade do que a da imagem, o que fez com que os métodos conhecidos de processamento de sinais não tivessem boa acurácia.

"Muitos sinais de interesse possuem menos informação do que a dimensão do ambiente sugere"[1] (CHEN; NEEDELL, 2015, p. 2, tradução livre). As formas tradicionais de aquisição de sinais baseadas no teorema de Nyquist (1) geram informação redundante. Tal fato traz a tona a pergunta: existe uma forma de aquisição de sinais na qual as amostras comprimidas são obtidas diretamente? (CHEN; NEEDELL, 2015). Em diversas situações, na verdade, o que se tem em mãos são somente as medidas que foram executadas, e conhecida a forma na qual a amostragem foi feita, além do conhecimento de que

[1] "Many signals of interest contain far less information than their ambient dimension suggests"

existe uma representação esparsa do sinal em determinada base. Em especial para essas situações o *compressed sensing* é bastante adequado.

O modelo de *compressed sensing* surgiu com esse intuito (DONOHO, 2006). Trabalhos na área mostram que, para certas classes de sinais, poucas amostras são necessárias para representar o sinal com acurácia (CHEN; NEEDELL, 2015).

De acordo com o modelo, para um dado sinal $\boldsymbol{f} \in \mathbb{C}^d$, amostragens são executadas na seguinte forma, utilizando produto interno:

$$y_i = \langle \boldsymbol{\phi_i}, \boldsymbol{f} \rangle \text{ para } i = 1,2,...,m, \tag{2.2}$$

na qual $m \ll d$. Os vetores $\boldsymbol{\phi_i} \in \mathbb{R}^d$ são linhas de uma matriz $\boldsymbol{\Phi} \in \mathbb{R}^{m \times d}$, chamada matriz de amostragem. Assim, pode-se reescrever o vetor de amostragem como $\boldsymbol{y} = \boldsymbol{\Phi f}$. Fica claro que, se $m \ll d$, reconstruir $\boldsymbol{f}$ a partir de $\boldsymbol{y}$, sem assumir nada a mais, é um problema com infinitas soluções (CHEN; NEEDELL, 2015).

Uma importante consideração que CS faz é a de que os sinais de interesse possuem menos informação do que sugere a dimensão d. Uma forma de quantificar essa noção é a esparsidade. Considere-se a quantidade de valores não nulos s de um sinal $\boldsymbol{f} \in \mathbb{C}^d$, também conhecida como norma ℓ_0:

$$\|\boldsymbol{f}\|_0 = s. \tag{2.3}$$

Considerando o mesmo sinal $\boldsymbol{f}$, diz-se que este é *s-esparso*, quando, para dado um um valor s, $\boldsymbol{f}$ satisfaz a seguinte inequação:

$$\|\boldsymbol{f}\|_0 \leq s \ll d. \tag{2.4}$$

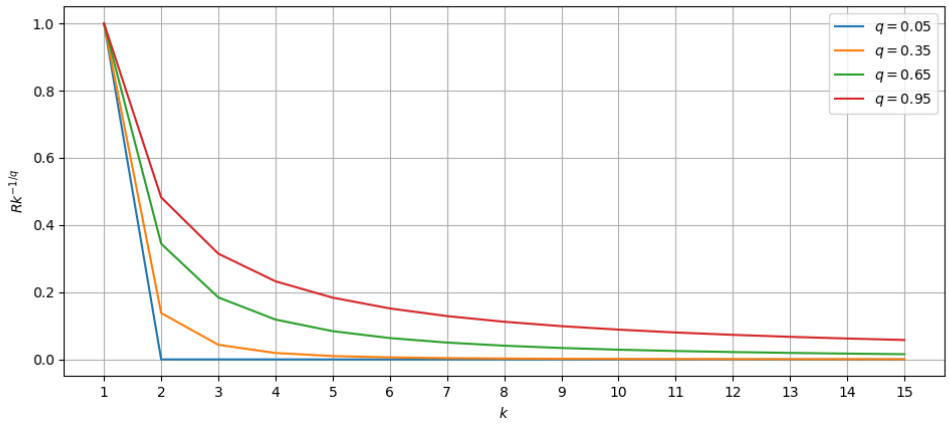
Na prática, sinais não são usualmente encontrados esparsos, o que leva a definição de *sinais compressíveis*, que são aqueles que obedecem à seguinte lei de decaimento exponencial (CHEN; NEEDELL, 2015):

$$|f_{\mathrm{k}}^*| < Rk^{-\frac{1}{q}}, \tag{2.5}$$

tal que $\boldsymbol{f}^*$ é o rearranjo decrescente de $\boldsymbol{f}$, ou seja, o vetor $\boldsymbol{f}$ reorganizado de forma que o maior coeficiente em magnitude seja f_1^*, o segundo maior, f_2^*, e assim por diante. R é uma constante positiva e $0 < q < 1$. Para ilustrar essa ideia, o gráfico abaixo apresenta curvas geradas variando q, usando $R = 1{,}0$ e no eixo x a variação de k. Um sinal compressível

é aquele que cada coeficiente f_k^* fica abaixo da curva apresentada, para algum q.

Figura 2.1: Curvas de limite superior para sinais compressíveis com $R = 1{,}0$, k no eixo x e valores diferentes valores de q.



Fonte: O Autor.

Percebe-se que para valores bem pequenos de q a compressibilidade se torna praticamente o mesmo que esparsidade. Segundo Chen and Needell (2015), se considerar-se $\boldsymbol{f_s}$ o vetor com os s maiores valores de $\boldsymbol{f}$ em magnitude, temos que, para sinais compressíveis $\boldsymbol{f}$ e $\boldsymbol{f_s}$:

$$\|\boldsymbol{f} - \boldsymbol{f_s}\|_2 \leq Rs^{\frac{1}{2}-\frac{1}{q}} \quad \text{e} \quad \|\boldsymbol{f} - \boldsymbol{f_s}\|_1 \leq Rs^{1-\frac{1}{q}}. \tag{2.6}$$

Por fim, a definição (2.4) exige que $\boldsymbol{f}$ seja esparso, ou seja, tenha s coeficientes não nulos no máximo. Por outro lado, $\boldsymbol{f}$ pode ser esparso em alguma outra base ortonormal $\mathbf{D}$, aqui referida como *sparsifying basis* (CANDES; TAO, 2005). Nesse caso, considera-se $\boldsymbol{f}$ s-esparso se:

$$\boldsymbol{f} = \mathbf{D}\boldsymbol{x} \quad \text{dado que} \quad \|\boldsymbol{x}\|_0 \leq s \ll d. \tag{2.7}$$

2.2.1 Mecanismos de Amostragem

A partir das definições básicas do modelo, pode-se formular o problema básico de CS da seguinte forma. Tomando um operador de amostragem $\boldsymbol{\Phi}$, o mapeamento linear de $\mathbb{C}^d$ para algum espaço de dimensão $\mathbb{C}^m$, para recuperarmos um sinal $\boldsymbol{f}$ a partir de suas medidas $\boldsymbol{y} = \boldsymbol{\Phi f}$, pode ser descrito como um problema de minimização:

$$\boldsymbol{f'} = \underset{\boldsymbol{g}\in\mathbb{C}^d}{\operatorname{argmin}} \|\boldsymbol{g}\|_0 \quad \text{sujeito a} \quad \boldsymbol{\Phi g} = \boldsymbol{y}. \tag{2.8}$$

Se $\boldsymbol{\Phi}$ não mapeia 2 vetores esparsos quaisquer para o mesmo valor de $\boldsymbol{y}$, ou seja, se $\boldsymbol{f} \neq \boldsymbol{g} \rightarrow \boldsymbol{\Phi f} \neq \boldsymbol{\Phi g}$, então a solução $\boldsymbol{f'}$ recupera $\boldsymbol{f}$, $\boldsymbol{f'} = \boldsymbol{f}$ (CHEN; NEEDELL, 2015). Esse problema, por outro lado, é intratável, e, em geral, NP-difícil (MUTHUKRISHNAN, 2005). Para que o operador de amostragem forneça o resultado anterior, ele deve ser *incoerente*. Dado um operador $\boldsymbol{\Phi}$ de colunas $\{\boldsymbol{\phi_i}\}$ com norma unitária, define-se a sua coerência μ como a maior correlação entre suas colunas:

$$\mu = \underset{i \neq j}{max} \langle \boldsymbol{\phi_i}, \boldsymbol{\phi_j} \rangle. \tag{2.9}$$

Fica definido, portanto, que um operador de amostragem é *incoerente* quando a sua coerência μ é suficientemente pequena. Por exemplo, um operador de amostragem que seja uma base ortonormal é incoerente. bem como operadores que sejam aproximadamente ortonormais para vetores esparsos. Nesse segundo caso, pode-se imaginar, por exemplo, que o conjunto de vetores esparsos utiliza somente os índices i, j e k. Dessa forma, apenas as colunas de $\boldsymbol{\Phi}$ correspondentes a i, j e k precisam ser ortonormais entre si, ou com baixa coerência entre essas mesmas colunas.

Uma propriedade que captura a mesma ideia que o referido acima foi desenvolvida por Candes and Tao (2006), chamada *restricted isometry property* (RIP). A constante de isometria restrita δ_s é o menor valor tal que:

$$(1 - \delta_s)\|\boldsymbol{f}\|_2^2 \leq \|\boldsymbol{\Phi f}\|_2^2 \leq (1 + \delta_s)\|\boldsymbol{f}\|_2^2 \quad \text{para todo vetor s-esparso } \boldsymbol{f}. \tag{2.10}$$

Diz-se que um operador de amostragem $\boldsymbol{\Phi}$ satisfaz a RIP de ordem s quando δ_s é suficientemente pequeno. A questão importante da RIP é descobrir qual o número de amostras m é necessária e quais são as classes de matrizes que possuem a propriedade. Dois importantes exemplos de matrizes que satisfazem a RIP são as seguintes (CHEN; NEEDELL, 2015):

- **Matrizes subgaussianas**: uma variável randômica $\boldsymbol{X}$ é subgaussiana se $\mathbb{P}(|\boldsymbol{X}| > t) \leq C_1 e^{-vt^2}$ para todo $t > 0$ e constantes positivas C_1 e v. Ou seja, variáveis subgaussianas possuem distribuição de média aproximadamente 0, e a maior parte dos valores próximos a média. Exemplos dessa distribuição são a própria distribuição Gaussiana e a matrizes de Bernoulli, que possuem coeficientes -1 e $+1$ distribuídos com média 0. Mendelson, Pajor and Tomczak-Jaegermann (2008) provaram que se $\boldsymbol{\Phi} \in \mathbb{R}^{m \times d}$ é subgaussiana, então, com alta probabilidade, $\frac{1}{\sqrt{m}}\boldsymbol{\Phi}$ satisfaz a RIP de ordem s quando m está na ordem de $s \log d$.

- **Matrizes parcialmente ortogonais limitadas**: dada uma matriz ortogonal $\mathbf{\Psi}$ com dimensões $d \times d$ cujos coeficientes são limitados por $\frac{C_2}{\sqrt{d}}$ para uma constante C_2. Uma matriz parcialmente ortogonal limitada $\mathbf{\Phi}$ pode ser obtida escolhendo m colunas de $\mathbf{\Psi}$ randomicamente. Por exemplo, a matriz da transformada discreta de Fourier tem coeficientes limitados em $\frac{1}{\sqrt{d}}$. Logo, escolhendo-se m linhas randomicamente tem-se uma matriz parcialmente ortogonal limitada. Rudelson and Vershynin (2008) provaram que essas matrizes satisfazem RIP com alta probabilidade quando o número de medidas m é na ordem de $s \log^4 d$.

2.2.2 Algoritmos aplicados ao modelo de Compressive Sensing

Ao considerar-se a formulação do problema geral de CS apresentada na seção anterior, um dos resultados também apresentados é de que o problema é NP-difícil. Sob essa ótica, diversas soluções alternativas foram desenvolvidas com o intuito de, através da solução de algum outro problema relacionado, ou do uso de alguma estratégia diferente, fosse possível chegar a uma solução ótima para a formulação de CS.

Inicialmente, é necessário levar em consideração que nem sempre as amostras coletadas estão completamente corretas. Dessa forma, tem-se um novo vetor de medidas, $\boldsymbol{y} = \mathbf{\Phi}\boldsymbol{f} + \boldsymbol{e}$, que considera o ruído de amostragem, ou vetor de erro, $\boldsymbol{e}$.

Além disso, é essencial definir as propriedades ideais de um método de recuperação baseado em CS. Segundo Chen and Needell (2015), as propriedades são as seguintes:

- **Amostragem não adaptativa:** Os operadoes de amostragem não devem ser dependentes do sinal. Operadores que possuem RIP também possuem essa propriedade.
- **Número ótimo de amostras:** O número de amostras necessárias deve ser mínimo.
- **Garatia de uniformidade:** Um único operador de amostragem deve ser suficiente para qualquer sinal.
- **Robustez:** O método deve ser estável e robusto em relação ao ruído, e possuir garantias quanto ao erro.
- **Complexidade:** O algoritmo deve ser computacionalmente eficiente.

Buscando obedecer o maior número das propriedades citadas acima, alguns métodos e algoritmos surgiram. Nas subseções seguintes serão descritas duas metodologias que foram utilizadas para tal.

2.2.2.1 Métodos baseados em otimização

Essa metodologia, utilizada pelos primeiros trabalhos na área, utiliza uma relaxação convexa como forma de chegar a solução do problema (2.8). Ou seja, utiliza a norma ℓ_1 ao invés da norma ℓ_0, de forma que o problema se torne convexo e solúvel através de métodos de programação linear.

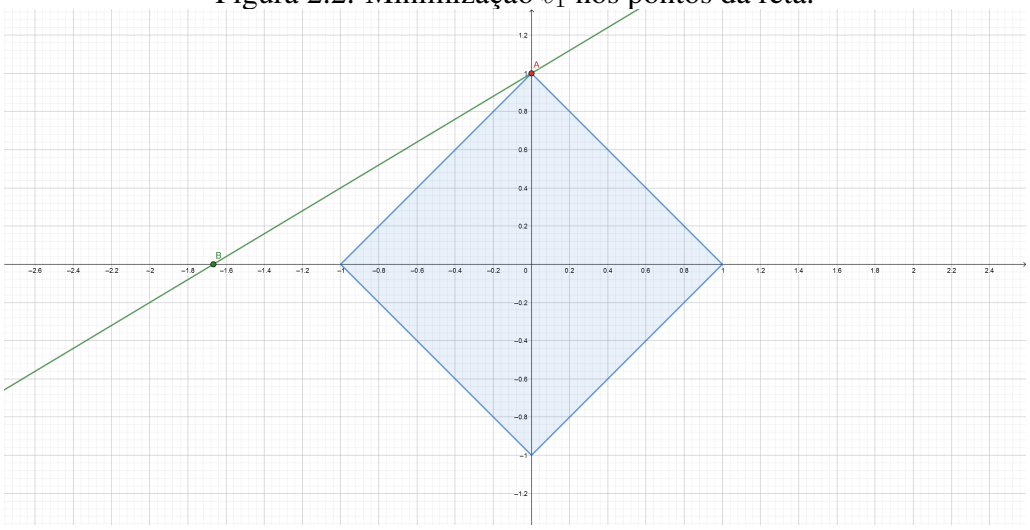
Dessa forma, o problema relaxado para a norma ℓ_1 fica como segue:

$$\boldsymbol{f'} = \underset{\boldsymbol{g}\in\mathbb{C}^d}{\operatorname{argmin}} \|\boldsymbol{g}\|_1 \quad \text{sujeito a} \quad \|\boldsymbol{\Phi g} - \boldsymbol{y}\|_2 \leq \epsilon, \tag{2.11}$$

tal que $\|\boldsymbol{e}\|_2 \leq \epsilon$.

A geometria da norma ℓ_1 permite esparsidade, como pode-se ver no exemplo a seguir. Imaginando-se que a reta em verde retrata todas as possíveis soluções para um sinal $\boldsymbol{f} \in \mathbb{R}^2$, os pontos mais esparsos da reta são $A = (0;1)$ e $B = (-1{,}66;0)$. A região em azul limita todos os pontos nos quais $\|\boldsymbol{p}\|_1 \leq 1{,}0$, tal que $\boldsymbol{p} \in \mathbb{R}^2$. Nesse caso, o ponto A é o de menor norma ℓ_1 da reta, sendo essa a solução do problema (2.11) e, ao mesmo tempo, uma das possíveis soluções do problema (2.8).

Figura 2.2: Minimização ℓ_1 nos pontos da reta.



Fonte: O Autor.

Candes, Romberg and Tao (2006b) chegaram a resultados quanto a garantias para os limites de erro, baseado na intensidade do ruído, como segue.

Teorema 2. *(CANDES; ROMBERG; TAO, 2006b). Dado um operador de amostragem* $\boldsymbol{\Phi}$ *que satisfaz a RIP. Então, para qualquer sinal* $\boldsymbol{f}$ *e sua amostragem ruidosa* $\boldsymbol{y} = \boldsymbol{\Phi f} + \boldsymbol{e}$*,*

tal que $\|\boldsymbol{e}\|_2 \leq \epsilon$*, a solução* $\boldsymbol{f'}$ *de* (2.11) *satisfaz:*

$$\|\boldsymbol{f'} - \boldsymbol{f}\|_2 \leq C_3 \left[\epsilon + \frac{\|\boldsymbol{f} - \boldsymbol{f_s}\|_1}{\sqrt{s}}\right],$$

tal que $\boldsymbol{f_s}$ *denota o vetor com os* s *maiores coeficientes em magnitude de* $\boldsymbol{f}$ *e* C_3 *é uma constante tal que* $C_3 \geq 0$*.* —

2.2.2.2 Métodos utilizando algoritmos gulosos

Depois da formalização do problema de CS, gerando a demanda para uma solução, surgiram os primeiros algoritmos que, de forma iterativa, resolvem o problema (2.8). Um dos principais algoritmos é o *Orthogonal Matching Pursuit (OMP)*, analisado por Gilbert e Tropp (TROPP; GILBERT, 2007). Segundo Tropp and Gilbert (2007), o algoritmo do OMP é o que segue:

Algoritmo 1: *Orthogonal Matching Pursuit (OMP).*

Entrada: Matriz de amostragem $m \times d$ $\boldsymbol{\Phi}$, Vetor de m medidas $\boldsymbol{y} = \boldsymbol{\Phi f}$, parâmetro de esparsidade s

Saída: Estimativa do sinal ideal $\boldsymbol{f'} \in \mathbb{R}^d$

$\boldsymbol{r_{(0)}} \leftarrow \boldsymbol{y}$;

$t \leftarrow 1$;

$\Lambda_{(0)} \leftarrow \emptyset$;

$\boldsymbol{\Phi_{(0)}} \leftarrow []$;

$\boldsymbol{f'} \leftarrow \boldsymbol{0}$;

while $t \leq s$ **do**

// Retorna o primeiro índice j que possui o maior produto interno.

$\lambda_{(t)} = \underset{j=1,\ldots,d}{argmax} |\langle \boldsymbol{r_{(t-1)}}, \boldsymbol{\phi_j} \rangle|$;

$\Lambda_{(t)} = \Lambda_{(t-1)} \cup \{\lambda_{(t)}\}$;

$\boldsymbol{\Phi_{(t)}} = \left[\boldsymbol{\Phi_{(t-1)}} \;\; \boldsymbol{\phi_{\lambda_{(t)}}}\right]$;

Resolver problema de mínimos quadrados: $\boldsymbol{x_{(t)}} = \underset{\boldsymbol{x}}{argmin} \|\boldsymbol{y} - \boldsymbol{\Phi_{(t)}} \boldsymbol{x}\|^2$;

$\boldsymbol{a_{(t)}} \leftarrow \boldsymbol{\Phi_{(t)}} \boldsymbol{x_{(t)}}$;

$\boldsymbol{r_{(t)}} \leftarrow \boldsymbol{y} - \boldsymbol{a_{(t)}}$;

$t \leftarrow t + 1$;

$\boldsymbol{f'}_{\Lambda_{(t-1)}} \leftarrow \boldsymbol{x_{(t-1)}}$;

Uma importante observação relativa ao sucesso do algoritmo OMP é que, dado que

a matriz $\boldsymbol{\Phi}$ é incoerente, seu conjugado transposto multiplicado por si mesmo é próximo à identidade, ou seja, $\boldsymbol{u} = \boldsymbol{\Phi}^*\boldsymbol{y} = \boldsymbol{\Phi}^*\boldsymbol{\Phi}\boldsymbol{f}$ é muito próximo a $\boldsymbol{f}$. Logo, o algoritmo OMP considera que o mais alto coeficiente de $\boldsymbol{u}$ é um dos coeficientes esparsos de $\boldsymbol{f}$ (CHEN; NEEDELL, 2015). Tropp and Gilbert (2007) ainda apresentam um teorema sobre o algoritmo.

Teorema 3. *(TROPP; GILBERT, 2007). Dada uma matriz $\boldsymbol{\Phi}$ de dimensões $m \times d$ sub-gaussiana de medidas tal que $m \geq Cs\log d$, e $\boldsymbol{f}$ um sinal s-esparso em $\mathbb{R}^d$. Então, com alta probabilidade, OMP reconstrui corretamente o sinal $\boldsymbol{f}$ a partir de suas medidas $\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{f}$.* —

Sem nenhuma modificação, o OMP não é reconhecidamente robusto a ruído, nem oferece garantias de uniformidade (vide Subseção 2.2.2). Porém, esse algoritmo possui um baixo custo computacional e boa eficiência, tendo complexidade $O(smd)$ (CHEN; NEEDELL, 2015). Algo que fica claro, observando-se o pseudo-código e a complexidade, é que o OMP tem desempenho inversamente proporcional à esparsidade da solução, ou seja, a cada nova iteração no *loop* principal, um novo coeficiente da solução é calculado.

Outros algoritmos de estratégia gulosa oferecem garantias de uniformidade, além de robustez a ruído, com limites demonstrados, bem como número de iterações e medidas. Dois principais destes algoritmos são o *CoSaMP (Compressive Sampling Matching Pursuit)* (NEEDELL; TROPP, 2009), e o *IHT (Iterative Hard Thresholding)* (BLUMENSATH; DAVIES, 2009).

2.2.2.3 Métodos utilizando Total Variation

Em diversas aplicações, os sinais de interesse são imagens. Imagens naturais normalmente são compressíveis em determinada base, como por exemplo Wavelets. Porém, a utilização dos métodos anteriores para recuperação destas imagens gera, algumas vezes, artefatos de alta frequência, que se tornam desagradáveis a visualização (CHEN; NEEDELL, 2015). Tal fato levou ao desenvolvimento de outro método de minimização, baseado em *Total Variation*, ou seja, na minimização da norma ℓ_1 do gradiente da imagem.

Para simplificar a definição, considere-se uma imagem em tons de cinza $\boldsymbol{X}$ de tamanho $n \times n$. Por definição, a *total variation norm* (TV) de uma imagem $\boldsymbol{X}$ é:

$$\|\boldsymbol{X}\|_{TV} = \sum_{j,k} \sqrt{\left(\boldsymbol{X}_{\boldsymbol{j+1,k}} - \boldsymbol{X}_{\boldsymbol{j,k}}\right)^2 + \left(\boldsymbol{X}_{\boldsymbol{j,k+1}} - \boldsymbol{X}_{\boldsymbol{j,k}}\right)^2} = \sum_{j,k} |(\boldsymbol{\nabla X})_{j,k}|, \quad (2.12)$$

no qual $\boldsymbol{\nabla X}$ é o gradiente da imagem (discreta) $\boldsymbol{X}$. O gradiente pode ser definido da seguinte forma:

$$(\boldsymbol{X_x})_{j,k} = \boldsymbol{X_{j+1,k}} - \boldsymbol{X_{j,k}}$$
$$(\boldsymbol{X_y})_{j,k} = \boldsymbol{X_{j,k+1}} - \boldsymbol{X_{j,k}},$$

resultando, por fim, em:

$$\boldsymbol{\nabla X} = \begin{cases} ((\boldsymbol{X_x})_{j,k}, (\boldsymbol{X_y})_{j,k}), & 1 \leq j \leq n-1, \;\; 1 \leq k \leq n-1 \\ (0, (\boldsymbol{X_y})_{j,k}), & j = n, \;\; 1 \leq k \leq n-1 \\ ((\boldsymbol{X_x})_{j,k}, 0), & 1 \leq j \leq n-1, \;\; k = n \\ (0,0), & j = n, \;\; k = n \end{cases} \tag{2.13}$$

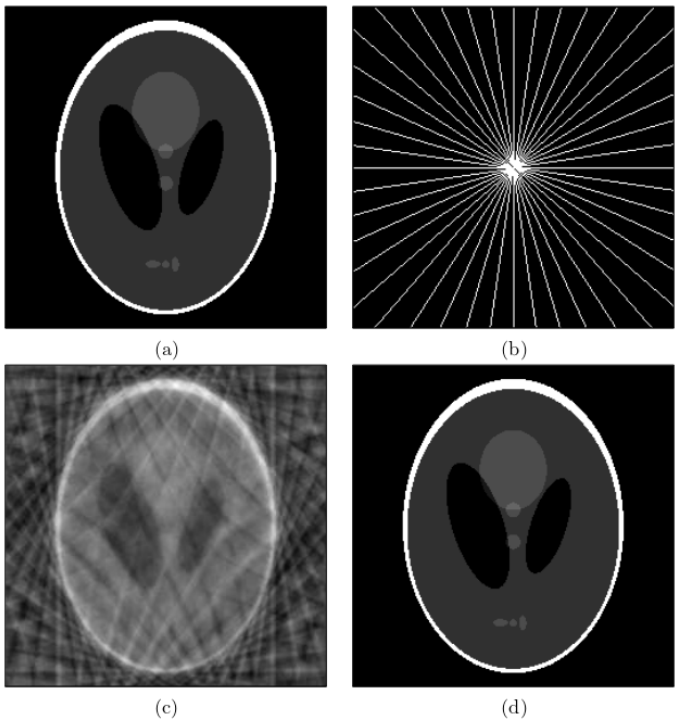
Após definidas essas bases, pode-se considerar a minimização a seguir, utilizando a norma TV (CHEN; NEEDELL, 2015).

$$\boldsymbol{X'} = \underset{\boldsymbol{M} \in \mathbb{R}^{n \times n}}{argmin} \|\boldsymbol{M}\|_{TV} \quad \text{sujeito a} \quad \|\boldsymbol{y} - A(\boldsymbol{M})\|_2 \leq \epsilon, \tag{2.14}$$

tal que A é uma função que realiza a vetorização da imagem, e em seguida amostragem de alguns coeficientes. $\boldsymbol{y} = A(\boldsymbol{X}) + \boldsymbol{e}$ são as medidas com ruído $\boldsymbol{e}$, sendo que $\|\boldsymbol{e}\|_2 \leq \epsilon$.

O ganho na representação de imagens se dá pelo fato de que não se cria uma representação esparsa da imagem diretamente, mas sim do gradiente da imagem, o que torna a mesma mais suave e remove artefatos com alta frequência. A norma TV foi utilizada em diversos trabalhos. Por exemplo, entre os primeiros resultados que motivaram o surgimento dessa área de estudo está o trabalho proposto por Candes, Romberg and Tao (2006a), no qual, possuindo aproximadamente 5% das medidas de imagens de ressonância magnética, foi possível recuperar toda a imagem, como é possível visualizar na figura abaixo.

Figura 2.3: Exemplo de recuperação de imagem de ressonância magnética utilizando a norma TV. (a) A imagem final correta desejada. (b) Amostragem que se possuía no domínio frequência. (c) Reconstrução via *Filtered BackProjection*. (d) Reconstrução utilizando a norma TV.



Fonte: Candes, Romberg and Tao (2006a)

2.2.3 Resultados com dicionários coerentes

A partir dessa seção, ao se tratar das matrizes de amostragem, se utilizará o termo dicionários, que representam uma transformação do sinal, não necessariamente uma transformação de domínio conhecida, podendo esta ser aprendida utilizando métodos que serão apresentados na seção 2.3. A utilização de matrizes de amostragens incoerentes, ou dicionários incoerentes, limita a aplicação de , *compressed sensing* para áreas cuja base de solução dos mesmos é baseada em dicionários incoerentes. Porém existem áreas com soluções baseadas em dicionários coerentes, que utilizam, por exemplo, dicionários supercompletos, com mais colunas que linhas. "Coerência é, de certa forma, uma propriedade natural para *compressed sensing*, porque se duas colunas são muito correlatas, será impossível em geral distinguir se a energia do sinal vem de uma ou da outra"[2] (CANDES et

[2] "Coherence is in some sense a natural property in the *compressed sensing* quadrowork, for if two columns are closely correlated, it will be impossible in general to distinguish whether the energy in the signal comes from one or the other"

al., 2011, tradução livre).

Porém, em alguns casos, não precisa-se recuperar os coeficientes originais do sinal, o que com dicionários coerentes não é possível, mas sim o próprio sinal já transportado para uma nova base. Matematicamente falando, imagine-se um sinal $\boldsymbol{f} = \mathbf{D}\boldsymbol{x}$, no qual suas medidas são $\boldsymbol{y} = \boldsymbol{\Phi f} = \boldsymbol{\Phi}\mathbf{D}\boldsymbol{x}$. Se o dicionário $\mathbf{D}$ for coerente, não é possível recuperar os coeficientes $\boldsymbol{x}$, mas é possível recuperar $\boldsymbol{f}$ (CANDES et al., 2011).

A partir desses estudos, Candes et al. (2011) apresentou uma nova definição para o problema de CS, baseado na solução utilizando a norma ℓ_1, considerando $\boldsymbol{f} = \mathbf{D}\boldsymbol{x}$ o sinal e $\boldsymbol{y} = \boldsymbol{\Phi f} + \boldsymbol{e}$ as medidas, $\boldsymbol{e}$ o erro.

$$\boldsymbol{f'} = \underset{\boldsymbol{g}\in\mathbb{C}^d}{\operatorname{argmin}} \|\mathbf{D}^*\boldsymbol{g}\|_1, \quad \text{sujeito a} \quad \|\boldsymbol{\Phi g} - \boldsymbol{y}\|_2 \leq \epsilon, \tag{2.15}$$

tal que $\|\boldsymbol{e}\|_2 \leq \epsilon$.

Candes et al. (2011) também redefiniu a RIP, chamando de D-RIP como segue. Dado um operador de amostragem $\boldsymbol{\Phi}$, ele satisfaz a D-RIP para um determinado dicionário $\mathbf{D}$ de ordem s se:

$$(1-\delta_s)\|\mathbf{D}\boldsymbol{x}\|_2^2 \leq \|\boldsymbol{\Phi}\mathbf{D}\boldsymbol{x}\|_2^2 \leq (1+\delta_s)\|\mathbf{D}\boldsymbol{x}\|_2^2 \quad \text{para todo vetor s-esparso, } \boldsymbol{x}. \tag{2.16}$$

e para algum δ_s de valor próximo a 0. Tomando a nova definição, Candes et al. (2011) provou o seguinte teorema sobre a recuperação do sinal.

Teorema 4. *(CANDES et al., 2011) Dado* $\mathbf{D}$ *um dicionário e supondo um operador de amostragem* $\boldsymbol{\Phi}$ *que satisfaz a D-RIP de ordem* s*, a solução* $\boldsymbol{f'}$ *para a análise* ℓ_1 *satisfaz*

$$\|\boldsymbol{f'} - \boldsymbol{f}\|_2 \leq C_4 \left[\epsilon + \frac{\|\mathbf{D}^*\boldsymbol{f} - (\mathbf{D}^*\boldsymbol{f})_s\|_1}{\sqrt{s}}\right],$$

tal que $(\mathbf{D}^*\boldsymbol{f})_s$ *denota o vetor com os* s *maiores coeficientes em magnitude de* $\mathbf{D}^*\boldsymbol{f}$ *e* C_4 *é uma constante tal que* $C_4 \geq 0$. —

A abordagem baseada em dicionários coerentes abriu caminhos para novas análises e usos de CS para outros problemas, além de tornar possível utilizar os modelos e técnicas de CS para outros campos do conhecimento.

2.3 Aprendizado de Dicionários

A área de aprendizado de dicionários, do inglês *dictionary learning*, busca, através dos dados já conhecidos de uma classe de sinais, prover novos dicionários nos quais os sinais serão capazes de se conformar de maneira mais esparsa. Sob o ponto de vista de CS, um dicionário $\boldsymbol{D}$ normalmente é escolhido de forma aleatória, desde que satisfaça a RIP. Aprendizado de dicionários, por outro lado, busca gerar modelos considerando os sinais envolvidos, ou seja, dicionários que dependem dos dados, ou *data-dependent dictionaries* (CHEN; NEEDELL, 2015).

Nas seções seguintes será explicitado o problema geral de aprendizado de dicionários, bem como um algoritmo que é capaz de de gerar um dicionário a partir de dados observados.

2.3.1 Problema Geral de Aprendizado de Dicionários

Nessa seção será apresentado um modelo geral do problema que aprendizado de dicionários busca resolver. De forma intuitiva, pode-se imaginar aprendizado de dicionários como se fosse desejado gerar um sub-dicionário da língua portuguesa a partir de uma grande quantidade textos.

Mais formalmente, seja $\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n} \in \mathbb{R}^L$ um conjunto finito de sinais de treinamento, e sejam m e s inteiros positivos. Deseja-se encontrar a matriz $\boldsymbol{D}$ e os vetores s-esparsos $\boldsymbol{\gamma_1}, \boldsymbol{\gamma_2}, ..., \boldsymbol{\gamma_n} \in \mathbb{R}^m$, tais que $\boldsymbol{D\gamma_i} \approx x_i$ para todo i. Dessa forma, pode-se definir o problema de aprendizado de dicionários utilizando a norma ℓ_2 como segue (CHEN; NEEDELL, 2015):

$$\underset{\boldsymbol{D},\boldsymbol{\gamma_1}...\boldsymbol{\gamma_n}}{min} \sum_{i=1}^{n} \|\boldsymbol{x_i} - \boldsymbol{D\gamma_i}\|_2^2 \quad \text{tal que} \quad \|\boldsymbol{\gamma_i}\|_0 \leq s, \text{ para todo } i. \tag{2.17}$$

Nesse caso, $\boldsymbol{D} \in \mathbb{R}^{L \times m}$ é o dicionário aprendido, enquanto cada vetor $\boldsymbol{\gamma_i} \in \mathbb{R}^m$, com no máximo s coeficientes não nulos, e $s \ll L$. Neste caso, um sinal $\boldsymbol{x_i}$ contendo L amostras é representado pela combinação linear de s colunas do dicionário $\boldsymbol{D}$. Os coeficientes desta combinação linear são armazenados nos s elementos não zero do vetor $\boldsymbol{\gamma_i}$. Ainda, sobre o problema (2.17), para que as escolhas de $\boldsymbol{\gamma_i}$ e $\boldsymbol{D}$ sejam únicas, pode-se forçar que as colunas de $\boldsymbol{D}$ tenham norma ℓ_2 unitária (CHEN; NEEDELL, 2015). O valor m é conhecido como o número de átomos do dicionário. É importante notar que a função

de erro de (2.17) usa norma ℓ_2, mas pode ser trocada por outra função de erro, como, por exemplo, a norma ℓ_1 (CHEN; NEEDELL, 2015).

Também é possível formular o problema com um dado erro de aproximação ϵ. Nesse caso, pode-se reformular o problema de aprendizado de dicionários como segue (CHEN; NEEDELL, 2015).

$$\underset{\boldsymbol{D},\boldsymbol{\gamma_1}\ldots\boldsymbol{\gamma_n}}{min} \sum_{i=1}^{n} \|\boldsymbol{\gamma_i}\|_0 \quad \text{tal que} \quad \|\boldsymbol{x_i} - \boldsymbol{D\gamma_i}\|_2 \leq \epsilon, \text{ para todo } i. \tag{2.18}$$

Unificando as duas formulações (2.17) e (2.18), chega-se ao seguinte (CHEN; NEEDELL, 2015):

$$\underset{\boldsymbol{D},\boldsymbol{\gamma_1}\ldots\boldsymbol{\gamma_n}}{min} \sum_{i=1}^{n} \|\boldsymbol{x_i} - \boldsymbol{D\gamma_i}\|_2^2 + \lambda\|\boldsymbol{\gamma_i}\|_0, \tag{2.19}$$

no qual λ é um parâmetro de balanceamento. Porém, utilizando-se a norma ℓ_0, o problema é usualmente NP-difícil. Dessa forma, pode-se gerar uma reformulação somente trocando a norma ℓ_0 pela norma ℓ_1 (CHEN; NEEDELL, 2015):

$$\underset{\boldsymbol{D},\boldsymbol{\gamma_1}\ldots\boldsymbol{\gamma_n}}{min} \sum_{i=1}^{n} \|\boldsymbol{x_i} - \boldsymbol{D\gamma_i}\|_2^2 + \lambda\|\boldsymbol{\gamma_i}\|_1 \tag{2.20}$$

Aprendizado de dicionários tem ligação com diversas outras áreas do conhecimento. Em especial com CS, da forma que foi explicitada no início da seção, e com processamento de imagens, o que será melhor abordados na seção Subseção 2.3.3.

2.3.2 *Online Dictionary Learning*

Um dos principais algoritmos de aprendizado de dicionários, com aplicação em processamento de imagens, é o algoritmo *Online Dictionary Learning (ODL)*. Um de seus principais diferenciais se dá pela possibilidade de utilizar um conjunto de treinamento bastante volumoso, mantendo bom desempenho (MAIRAL et al., 2009). Por isso, pode-se dizer que o ODL possui boa escalabilidade.

O ODL é um algoritmo *online*, ou seja, a cada nova iteração t é gerada uma nova versão do dicionário $\boldsymbol{D_{(t)}}$ a partir de uma nova amostra $\boldsymbol{x_{(t)}}$.

Entrando mais a fundo na definição geral do algoritmo, tomado um chute inicial $\boldsymbol{D_{(0)}}$, as próximas iterações $t \geq 1$ serão baseadas em $\boldsymbol{D_{(t-1)}}$ e $\boldsymbol{x_{(t)}}$. O novo dicionário $\boldsymbol{D_{(t)}}$ será gerado a partir de dois passos. Primeiramente, deve-se encontrar o vetor de

coeficientes $\boldsymbol{\gamma_{(t)}}$ relativos a $\boldsymbol{x_{(t)}}$ e $\boldsymbol{D_{(t-1)}}$, resolvendo o problema de codificação esparsa a seguir (CHEN; NEEDELL, 2015):

$$\boldsymbol{\gamma_{(t)}} = \underset{\boldsymbol{\gamma}}{argmin} \frac{1}{2} \|\boldsymbol{x_{(t)}} - \boldsymbol{D_{(t-1)}}\boldsymbol{\gamma}\|_2^2 + \lambda \|\boldsymbol{\gamma}\|_1, \tag{2.21}$$

sendo que λ é um parâmetro de configuração do algoritmo. Esse problema pode ser resolvido por algum dos algoritmos discutidos na Subseção 2.2.2. Também é possível substituir a norma ℓ_1 pela ℓ_0, de forma que algum algoritmo guloso de CS possa ser utilizado. O mesmo se dará para o passo seguinte.

No segundo passo do algoritmo, definimos o novo dicionário $\boldsymbol{D_{(t)}}$ através dos vetores de coeficientes anteriores, $\boldsymbol{\gamma_{(1)}}, ..., \boldsymbol{\gamma_{(t-1)}}$, além de considerar o novo $\boldsymbol{\gamma_{(t)}}$. Para facilitar, consideremos a matriz $\boldsymbol{X}$ com colunas $\boldsymbol{x_1}$,...,$\boldsymbol{x_t}$ e a matriz $\boldsymbol{\Gamma}$ com colunas $\boldsymbol{\gamma_1}$,...,$\boldsymbol{\gamma_t}$. Dessa forma, e considerando que os valores de $\boldsymbol{\gamma_i}$ são fixos, pode-se formular o problema da seguinte forma (CHEN; NEEDELL, 2015):

$$\boldsymbol{D_{(t)}} = \underset{\boldsymbol{D}}{argmin} \frac{1}{t} \sum_{i=1}^{t} \|\boldsymbol{x_i} - \boldsymbol{D}\boldsymbol{\gamma_i}\|_2^2. \tag{2.22}$$

A Equação 2.22 pode ser resolvida utilizando-se o método do gradiente descendente. Para isso, primeiramente precisamos definir as seguintes matrizes (CHEN; NEEDELL, 2015).

$$\boldsymbol{A_{(t)}} = \sum_{i=1}^{t} \boldsymbol{\gamma_i}\boldsymbol{\gamma_i^T} \in \mathbb{R}^{m \times m}, \quad \boldsymbol{B_{(t)}} = \sum_{i=1}^{t} \boldsymbol{x_i}\boldsymbol{\gamma_i^T} \in \mathbb{R}^{L \times m}. \tag{2.23}$$

Tomando como valor inicial $\boldsymbol{D} = \boldsymbol{D_{(t-1)}}$, e utilizando o seguinte cálculo recursivo em cada coluna $\boldsymbol{d_k}$, que sempre minimiza a função objetivo (MAIRAL et al., 2009), chega-se no novo dicionário $\boldsymbol{D_{(t)}}$, iterando até que não se tenha mudança em $\boldsymbol{D}$.

$$\boldsymbol{d_k} \leftarrow \frac{1}{\boldsymbol{A_{(t)_{k,k}}}} \left(\boldsymbol{B_{(t)_{:,k}}} - \boldsymbol{D}\boldsymbol{A_{(t)_{:,k}}}\right) + \boldsymbol{d_k}. \tag{2.24}$$

Por fim, será descrito o algoritmo do ODL, conforme Mairal et al. (2009).

Algoritmo 2: *Online Dictionary Learning (ODL).*

Entrada: Distribuição de probabilidade $p(\boldsymbol{x})$, para $\boldsymbol{x} \in \mathbb{R}^L$, parâmetro λ, dicionário inicial $\boldsymbol{D_{(0)}}$, número de iterações T.

Saída: Dicionário final $\boldsymbol{D_{(t)}}$.

$\boldsymbol{A_{(0)}} \leftarrow \boldsymbol{0}$;

$\boldsymbol{B_{(0)}} \leftarrow \boldsymbol{0}$;

for $t = 1..T$ **do**

- Escolher nova amostra $\boldsymbol{x_{(t)}}$ de $p(\boldsymbol{x})$;
- Resolver: $\boldsymbol{\gamma_{(t)}} = \underset{\gamma}{argmin} \|\boldsymbol{x_{(t)}} - \boldsymbol{D_{(t-1)}}\boldsymbol{\gamma}\|_2^2 + \lambda\|\boldsymbol{\gamma}\|_1$;
- $\boldsymbol{A_{(t)}} \leftarrow \boldsymbol{A_{(t-1)}} + \boldsymbol{\gamma_{(t)}}\boldsymbol{\gamma_{(t)}^T}$;
- $\boldsymbol{B_{(t)}} \leftarrow \boldsymbol{B_{(t-1)}} + \boldsymbol{x_{(t)}}\boldsymbol{\gamma_{(t)}^T}$;
- $\boldsymbol{D} \leftarrow \boldsymbol{D_{(t-1)}}$;
- Realizar a atualização do dicionário $\boldsymbol{D}$ para todas as colunas k até convergência: $\boldsymbol{d_k} \leftarrow \frac{1}{\boldsymbol{A_{(t)_{k,k}}}}\left(\boldsymbol{B_{(t)_{:,k}}} - \boldsymbol{D}\boldsymbol{A_{(t)_{:,k}}}\right) + \boldsymbol{d_k}$;
- **if** $\|\boldsymbol{d_k}\|_2 > 1$ **then**
  - Normalizar $\boldsymbol{d_k}$;
- $\boldsymbol{D_{(t)}} \leftarrow \boldsymbol{D}$;

return $\boldsymbol{D_{(t)}}$

2.3.3 Codificação Esparsa

A partir das bases de *compressed sensing* e de aprendizado de dicionários, pode-se buscar um outro objetivo, compressão de sinais. No caso que será estudado, o foco será em codificação esparsa de imagens, buscando embasar os estudos para a proposta do trabalho em questão.

É necessário considerar que imagens não são usualmente esparsas no domínio tempo. Porém, no domínio da transformada do cosseno, ou no domínio da DCT (*Discrete Cosine Transform*), imagens são, em geral, compressíveis, vide (2.5), fato que permite, por exemplo, gerar representações da imagem com alta fidelidade, mesmo utilizando baixa quantidade de informação.

Por outro lado, pode-se imaginar que, para uma determinada classe de imagens, por exemplo, imagens naturais, a utilização de um dicionário previamente treinado pode

gerar bons resultados. Para remoção de ruído em imagens, Elad and Aharon (2006) mostram que o uso de dicionários aprendidos têm resultados melhores que o uso de dicionários gerais, teóricos.

Considerando uma imagem I, separada em patches p_i de tamanho $\sqrt{L} \times \sqrt{L}$, com ou sem sobreposição [3] e vetorizando cada patch p_i em vetores $\boldsymbol{x_i} \in \mathbb{R}^L$. Tomando um dicionário $\boldsymbol{D} \in \mathbb{R}^{L \times m}$, aprendido via patches de imagens correlatas a I, se desejarmos representar os patches $\boldsymbol{x_i}$ em relação a $\boldsymbol{D}$, podemos utilizar a seguinte formulação:

$$\boldsymbol{\gamma_i} = \underset{\gamma}{argmin} \|\boldsymbol{x_i} - \boldsymbol{D\gamma}\|_2^2 \quad \text{sujeito a} \quad \|\boldsymbol{\gamma}\|_0 \leq s \text{ para todo } i, \tag{2.25}$$

tal que s é o número máximo de valores não nulos da representação $\boldsymbol{\gamma_i} \in \mathbb{R}^m$. Nesse caso, pode-se utilizar algum dos algoritmos apresentados na Seção 2.2.2.2, por exemplo o OMP. Para que a representação seja esparsa, precisa-se garantir que $s \ll L$. Por exemplo, tomando uma imagem I e separando em patches de 8×8, ou seja, $L = 64$, poderia ser escolhido um valor de $s = 4$ ou $s = 6$, independente do valor de m, garantido $s \leq m$, cujo valor escolhido, por exemplo, poderia ser $m = 256$.

Existem diversas outras aplicações de aprendizado de dicionários no campo de processamento de imagens, principalmente (MAIRAL; BACH; PONCE, 2014):

- *Denoising*: remoção de ruído de imagens;
- *Inpainting*: preenchimento de regiões não *pintadas* de imagens;
- *Demosaicking*: reconstrução da imagem a partir de sensores digitais;
- *Up-scaling*: aumento da resolução de imagens.

Estas aplicações fogem ao escopo do trabalho apresentado e, portanto, não serão aprofundadas nesse referencial, porém são amplamente desenvolvidas por diversos autores da área.

[3] Normalmente, para problemas de remoção de ruído, é utilizada sobreposição, enquanto para representação esparsa não é utilizada.

3 TRABALHOS RELACIONADOS

Neste capítulo serão abordados alguns trabalhos que se relacionam com a proposta apresentada. O foco principal da proposta está na codificação esparsa de quadros de vídeos utilizando dicionários previamente treinados, buscando executar o processo em tempo real.

Todos os trabalhos encontrados, bem como os padrões de vídeo existentes, utilizam a técnida de separar os quadros em blocos, ou *patches*, para que a aplicação das técnicas de transformação e compressão sejam mais eficientes.

Bryt and Elad (2008), em seu trabalho, fizeram a compressão de imagens de faces, utilizando o algoritmo K-SVD para aprendizado de um dicionário. Cada imagem facial foi amostrada para as mesmas dimensões, com as faces centralizadas em um mesmo ponto, de forma que, aproximadamente, cada componente da face localiza-se em uma mesma posição. Por fim, cada imagem foi dividia e patches de tamanho 15×15 sem sobreposição. Para cada posição de patch, foi treinado um dicionário de 512 átomos. A partir dessas definições, Bryt and Elad (2008) chegaram a resultados com alta taxa de compressão, e ótimas qualidades de recuperação da imagem. Os resultados alcançados pelos autores indicam a possibilidade de se utilizar essa metodologia de forma menos restritiva, com, possivelmente, resultados menos impressionantes tanto em qualidade quanto em compressão.

Alguns trabalhos utilizam as teorias de CS para enviarem amostragens dos blocos dos vídeos por parte do *encoder*, utilizando-se dos teoremas provados pela teoria. Prades-Nebot, Ma and Huang (2009) apresentou um trabalho exatamente nesse sentido. Seguindo o mesmo caminho, e buscando aprimorar a técnica apresentada por PRADES-NEBOT; MA; HUANG, Do et al. (2009) propõe uma mescla entre os métodos tradicionais de codificação de vídeos e a utilização de CS, conseguindo bons resultados e abrindo horizontes nesse sentido. Em ambos trabalhos, a etapa de CS deve ser executada por parte do *decoder*, o que é bastante custoso do ponto de vista de vídeos de maior resolução.

Zhang et al. (2008) propõe um método que também mescla os métodos existentes com a ideia de enviar amostragens. No trabalho proposto, apresenta uma forma de definir como cada bloco será codificado, se utilizando amostragem ou DCT. Os blocos que utlizam amostragem executam o processo de minimização da norma TV, enquanto os demais blocos aplicam a DCT inversa, por parte do *decoder*. A técnica foi incorporada a codificação H.264, chegando a ganhos significativos na codificação (ZHANG et

al., 2008).

Além destes trabalhos, o uso de CS para aprimorar vídeos de baixa qualidade foi apresentado por Xiang and Cai (2011), com foco em distribuição de vídeos para redes Wireless. O uso de CS, nesse caso, se dá com o intuito de, além de melhorar os vídeos de baixa qualidade e *bitrate*, fornecer um canal capaz de lidar com ruído, tal qual a teoria de CS prevê.

Todos os trabalhos citados anteriormente utilizam como dicionário a DCT. No sentido de utilizar técnicas de aprendizado de dicionários, Chen, Kang and Lu (2010) apresenta uma proposta baseada na de Zhang et al. (2008), porém, por parte do *decoder*, ao invés de utilizar algum dicionário conhecido, utilizar um aprendido.

Lima et al. (2012) apresenta em seu trabalho um codec de vídeo bastante próximo ao proposto. Em sua proposta, o autor utiliza o algoritmo K-SVD para treinamento de um dicionário de 256 átomos, seguindo a proposta de BRYT; ELAD, conforme descrito anteriomente. Para a decomposição, também utiliza o algoritmo OMP, utilizando blocos 8×8, sem sobreposição. As principais diferenças entre o trabalho descrito e o proposto estão no uso do ODL como algoritmo de treinamento de dicionários, além da forma como são processados os blocos. No trabalho descrito, LIMA et al. utiliza a diferença entre o bloco anterior e o atual ou somente o atual, dado um limite. No trabalho proposto, sempre será utilizado o bloco atual. Porém, quando a diferença não for significativa, não recalculará a representação do bloco.

No próximo capítulo será detalhada a proposta do trabalho, explicitando detalhes da implementação. Em seguida, testes e resultados serão apresentados com base na proposta.

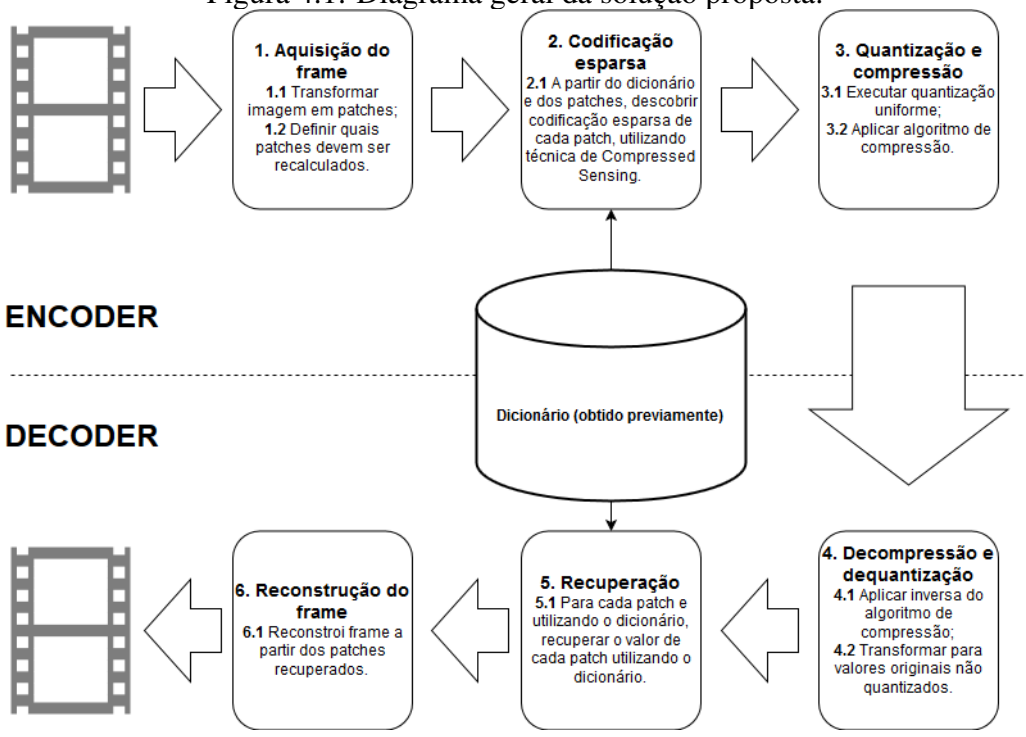
4 APLICAÇÃO DE TÉCNICAS DE COMPRESSED SENSING E APRENDIZADO DE DICIONÁRIOS PARA CODIFICAÇÃO DE VÍDEOS EM TEMPO REAL

Após considerar o desenvolvimento da área de CS e de aprendizado de dicionários e resumidos alguns trabalhos que focaram em alguma dessas áreas, será descrita a proposta do trabalho de forma detalhada.

O trabalho proposto visa desenvolver uma metodologia para codificação de vídeos em tempo real, utilizando dicionários pré-aprendidos relacionados ao próprio vídeo. Por ser um *quadrowork* com foco em tempo real, transmissão e recepção, naturalmente se põe em questão a necessidade de definir as funções que serão desenvolvidas por parte do *encoder* e por parte do *decoder*.

A imagem a seguir apresenta um diagrama geral de blocos, dividido entre *encoder* e *decoder*, além de representar blocos funcionais de cada uma das partes de forma geral.

Figura 4.1: Diagrama geral da solução proposta.



Fonte: O Autor.

Entende-se que os quadros processados estarão no formato de imagem colorida, pixel a pixel, RGB (Red, Green, Blue). Ou seja, os quadros terão dimensão $m \times n \times 3$, sendo a resolução do vídeo $m \times n$. No caso do sistema criado, a ordem dos pixels é linha a linha, ou seja, primeiro a linha 0 e todas suas colunas, depois a 1, e assim por diante. Em

contraste com essa definição o quadro em memória poderia ser coluna por coluna, o que alteraria alguns cálculos internos, sendo possível adaptar o sistema desenvolvido de forma bastante simples. Essa é uma diferença para com os codecs conhecidos, que utilizam comumente o espaço de cor YCbCr para processamento. O codec proposto não necessita fazer tal transformação, pois a transformação executada pelas técnicas de aprendizado de dicionários absorve a transformação no espaço de cor. Em seguida são definidas as nomenclaturas de alguns parâmetros importantes da abordagem proposta.

4.1 Parâmetros do sistema

Os parâmetros modificáveis do sistema proposto podem ser vistos na Tabela 4.1. As escolhas e resultados obtidos com a alteração dos parâmetros serão discutidas no Capítulo 5.

Tabela 4.1: Parâmetros do sistema.

Parâmetro	*Descrição*
p	Tamanho do patch.
K	Número de átomos do dicionário, também descrito como "tamanho"do dicionário.
s	Valor máximo da norma ℓ_0 de cada patch. Quantidade de valores não nulos da representação do patch.
q	Parâmetro de esparsidade de envio da informação, $q \leq s$.

Fonte: O Autor.

Nas seções seguintes serão descritas mais a fundo cada uma das etapas, seguindo a ordem apresentada na Figura 4.1, além de explicitar a ligação entre cada um dos blocos funcionais do projeto.

4.2 Aprendizado de dicionário

A etapa de aprendizado do dicionário, a qual é realizada off-line, é essencial para que a qualidade da representação esparsa utilizada seja boa. Além disso, para o caso de treinamento a partir de quadros de vídeos, caso em que se está trabalhando, é também necessário fazer a escolha de um algoritmo que permita a utilização de uma grande quantidade de dados de treinamento.

Considerando esses aspectos e o desempenho apresentado por Mairal et al. (2009), foi escolhido como algoritmo de treinamento do dicionário o *Online Dictionary Learning*

(MAIRAL et al., 2009), descrito na Subseção 2.3.2. Este algoritmo apresenta bom resultado para grande número de sinais de treinamento.

O treinamento foi executado da seguinte forma:

1. Os vídeos foram separados em diversas classes. Por exemplo, vídeos naturais, vídeos de esportes, *cartoon*, etc.;
2. Os quadros foram sorteados, de forma a garantir que um comportamento dos últimos quadros não prepondere sobre todo o dicionário;
3. Cada quadro foi separado em patches com tamanho $p \times p$; Para o treinamento, foram utilizados patches com sobreposição. Por fim, todos os patches foram vetorizados como descritos na Seção 4.3;
4. O dicionário foi obtido executando o algoritmo ODL (Algoritmo 2).

Vale ressaltar que, para vídeos muito especializados, pode-se utilizar um treinamento mais especializado. Por exemplo, uma transmissão esportiva, de um esporte específico pode se beneficiar de um dicionário obtido a partir de apenas quadros desse determinado esporte.

A fase de aprendizado do dicionário, por envolver um tempo maior de processamento, deve ser executada antes da transmissão em tempo real, por parte do *encoder*. Isso é possível, mesmo que não se conheça a transmissão em si, mas se conheça que tipo de conteúdo será transmitido.

Um importante parâmetro que deve ser escolhido é o número de átomos (i.e., colunas do dicionário) K do dicionário. Escolhas de valores maiores tendem a proporcionar maior qualidade na representação do vídeo, porém com perda de desempenho. No Capítulo 5 serão mostrados resultados de escolhas diferentes no número de átomos do dicionário.

Como parametrização do ODL, utiliza-se o valor s como esparsidade desejada e K como número de átomos do dicionário. O algoritmo ainda precisa saber o número de amostras que devem ser selecionadas aleatoriamente de cada quadro. Para determinar isso, foram executados testes de qualidade com diversos valores (Seção 5.2). Foi utilizada a implementação da biblioteca SPAMS (MAIRAL, 2009) do algoritmo ODL.

O resultado final desse etapa será um dicionário $\boldsymbol{D} \in \mathbb{R}^{3p^2 \times K}$, que será utilizado nas etapas seguintes de codificação e decodificação. Assim, portanto, em caso de uma transmissão de tempo real ou até mesmo se o vídeo for salvo em arquivo, uma das primeiras informações que deve ser enviada é o dicionário que servirá de base para a

transmissão.

4.3 Aquisição do quadro

Essa etapa tem como entrada o quadro a ser codificado, sendo que o histórico de quadros já processados, para a posterior verificação de igualdade, está armazenado também neste subsistema.

Já nesse ponto do processamento é necessário ter em vista o tamanho dos patches $p \times p$ que será utilizado. A escolha deve seguir o parâmetro definido na escolha do dicionário. Valores típicos a serem escolhidos são $p = 4$ ou $p = 8$. Na seção 5.2 serão discutidos resultados alcançados com ambas escolhas de tamanho de patch.

A escolha por processar patches se dá pela necessidade do processamento em tempo real, pois os algoritmos relacionados a CS dependem diretamente do tamanho dos sinais envolvidos, do tamanho dos dicionários e do número de valores não nulos. Porém, o tamanho dos dicionários e o número de valores não nulos precisam ser maiores quanto maior for o sinal, o que leva a um aumento de complexidade maior que linear com o aumento do tamanho dos patches, além de uma utilização maior de memória para o dicionário. Utilizar patches pequenos possibilita a paralelização dos cálculos. No caso do trabalho que está sendo apresentado, em que o processamento dos patches, que será comentado na próxima seção, se dá utilizando a GPU, a utilização de maior paralelismo tende a aumentar a velocidade de processamento. Resultados nesse sentido serão discutidos na Seção 5.2. Como já comentado na Seção 4.2, existe também um *trade-off* entre diminuir o tamanho do patch e aumentar a compressão.

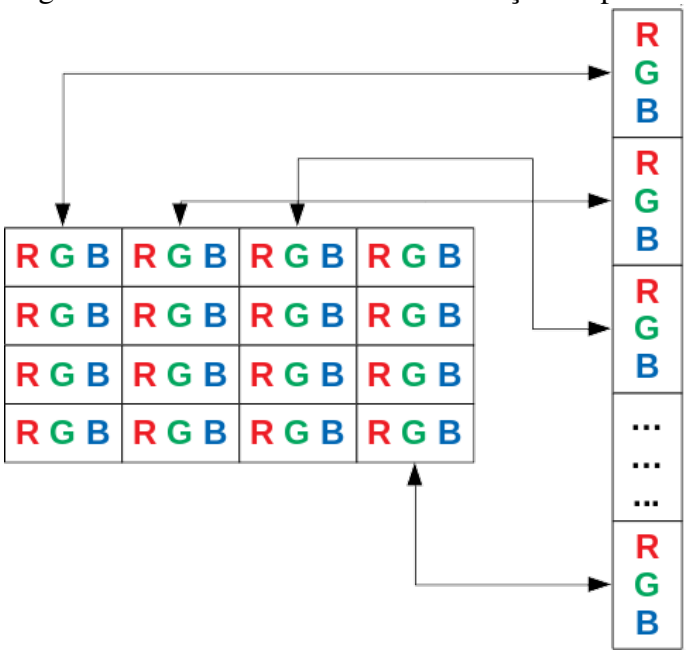
Tomada a imagem RGB $\boldsymbol{I}$ de dimensões $m \times n \times 3$, ela será separada em patches $p \times p \times 3$, sem sobreposição. Por exemplo, um dos patches iniciará em $\boldsymbol{I}_{\mathbf{0,0}}$ e terá parte em p linhas e p colunas, indo a $\boldsymbol{I}_{\boldsymbol{p-1,p-1}}$. Em seguida, o próximo patch iniciará em $\boldsymbol{I}_{\mathbf{0},\boldsymbol{p}}$ e terminará em $\boldsymbol{I}_{\boldsymbol{p-1,2p-1}}$. Nos finais de linha e de coluna, os valores de m e n podem não ser divisíveis por p, o que gera um resíduo. Nesse caso, foi adotada a solução que o padrão JPEG utiliza de repetir o último pixel a esquerda para as colunas e o pixel acima para as linhas, até que chegue-se a um patch $p \times p$ (MITCHELL, 1992). Junto ao processo de tomada dos patches, estes são vetorizados, também no formato linha a linha, para cada pixel RGB. Para um patch $\boldsymbol{P}$, os três primeiros valores do vetor corresponderão ao RGB de $\boldsymbol{P}_{\mathbf{0,0}}$. Os próximos três a $\boldsymbol{P}_{\mathbf{0,1}}$, e assim por diante. Como resultado final, pode-se dizer

que a imagem $\boldsymbol{I}$ foi separada em

$$T = \left\lceil \frac{m}{p} \right\rceil \left\lceil \frac{n}{p} \right\rceil$$

vetores de dimensão $L = 3p^2$. Pode-se escrever esses vetores como uma matriz $\boldsymbol{X} \in \mathbb{R}^{L \times T}$, na qual cada coluna $\boldsymbol{x_i}$ da matriz representa um dos patches vetorizados. A Figura 4.2 ilustra o processo de vetorização de um patch.

Figura 4.2: Funcionamento da vetorização do patch.



Fonte: O Autor.

Além do processo de vetorização, este subsistema também verifica se o patch atual i é igual ao patch i correspondente no quadro anterior. Caso seja, marca um vetor de booleanos $\boldsymbol{c} \in \mathbb{B}^T$, que indica os patches que precisam ou não ser recalculados. A métrica considera que um patch deve ser recalculado caso qualquer um dos valores dos pixels inseridos nele mudem. Esse simples cálculo sobre os patches garante que, para vídeos estáticos, o desempenho da aplicação seja acelerado. Parte considerável dos vídeos possuem partes com essas características, e esse cálculo simples não aumenta significativamente a complexidade do sistema como um todo.

4.4 Codificação esparsa dos patches

A partir dos patches $\boldsymbol{X}$ da etapa anterior, do dicionário $\boldsymbol{D}$, e do parâmetro de esparsidade desejada s, pode-se formular o problema de codificação esparsa, tal que $\boldsymbol{\gamma_i}$ seja a representação esparsa de $\boldsymbol{x_i}$, como segue:

$$\boldsymbol{\gamma_i} = \underset{\boldsymbol{\gamma}}{argmin} \|\boldsymbol{x_i} - \boldsymbol{D\gamma}\|_2^2 \quad \text{sujeito a} \quad \|\boldsymbol{\gamma}\|_0 \leq s, \text{ para todo } i. \tag{4.1}$$

O problema acima, como mencionado anteriormente, pode ser resolvido via OMP. A implementação básica escolhida para o sistema foi baseada na biblioteca SPAMS (MAIRAL, 2009). A partir dela, foi desenvolvida uma versão em CUDA®, focada em paralelizar o algoritmo OMP para cada $(\boldsymbol{x_i}, \boldsymbol{\gamma_i})$. Assim, foi possível obter uma representação em tempo real, conforme será verificado no Capítulo 5.

Um importante fato a ser considerado sobre o Algoritmo 1 (Subseção 2.2.2.2) é que, a cada iteração i, ele descobre a melhor solução para o parâmetro de esparsidade i. Ou seja, se escolhido $s = 5$, por exemplo, é possível armazenar-se os resultados com parâmetro de esparsidade $s = 1,...,5$. A utilidade desse fato se dá no controle da taxa de transmissão, pois quanto mais esparsa a representação, menor o bitrate. Portanto, com a aplicação do OMP, é possível gerar as melhores e piores representações. Imaginando múltiplas transmissões a partir do mesmo *encoder*, cada uma poderia ter uma taxa diferente sem a necessidade de mais cálculos nessa etapa. Assim, montamos para cada patch i uma matriz de representação $\boldsymbol{\Gamma_i} \in R^{K \times s}$ que armazena as várias representações esparsas para o patch i. Cada coluna $(\boldsymbol{\Gamma_i})_j$ é j-esparsa, onde j corresponde aos valores de s de 1 a s_max (no exemplo anterior $s_max = 5$). K corresponde ao número de átomos de $\boldsymbol{\gamma_i}$, i.e., o número de elementos do vetor $\boldsymbol{\gamma_i}$.

4.5 Quantização e compressão

A partir da codificação esparsa apresentada na seção anterior, a ideia dessa seção é fazer com que a codificação seja a mais comprimida possível, sem perdas sensíveis na representação do sinal. Além disso, ao final dessa seção, é necessário que haja uma representação em formato de *stream* de bytes, para ser enviada ou armazenada pelo *encoder*.

Primeiramente, o sinal é quantizado uniformemente em 8 bits. Para isso, é necessário identificar-se os coeficientes máximo u e mínimo l em $\boldsymbol{\Gamma}$. Em seguida, cada

coeficiente c de $\mathbf{\Gamma}$ é quantizado de acordo com a seguinte equação:

$$c' = \left\lfloor 0{,}5 + 255\frac{c-l}{u-l} \right\rfloor.$$

Um dos parâmetros que entra em ação é o parâmetro de esparsidade de envio da informação, q. Ele é o responsável pela escolha de qual será o nível de esparsidade escolhido, dado que $1 \leq q \leq s$. A partir desse parâmetro escolhe-se a coluna q de cada $\mathbf{\Gamma_i}$ (já quantizada), formando $\boldsymbol{C} \in \mathbb{N}^{2q \times T}$. Para representação de $\boldsymbol{C}$ são utilizadas as matrizes $\boldsymbol{V} \in \mathbb{N}^{q \times T}$ e $\boldsymbol{R} \in \mathbb{N}^{q \times T}$, sendo que $\boldsymbol{V}$ representa os coeficientes não nulos de $\boldsymbol{C}$ e $\boldsymbol{R}$ representa os índices dos mesmos:

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{V} \\ \boldsymbol{R} \end{bmatrix}.$$

Para geração do *bitstream*, são criados vetores a partir de $\boldsymbol{V}$ e $\boldsymbol{R}$. O exemplo a seguir ilustra uma situação para a qual $q = 3$ e o número de átomos em $\boldsymbol{\gamma_i}$ é $K = 16$. K limita o maior valor que os índice da matriz $\boldsymbol{R}$ podem assumir.

$$\boldsymbol{V} = \begin{bmatrix} 189 & 36 & 114 \\ 45 & 0 & 32 \\ 8 & 0 & 0 \end{bmatrix}, \boldsymbol{R} = \begin{bmatrix} 7 & 12 & 15 \\ 1 & 0 & 3 \\ 5 & 0 & 0 \end{bmatrix}$$

tal que as entradas com valor 0 correspondem a entradas que não foram necessárias (i.e., a representação esparsa do patch correspondente à respectiva coluna das matrizes contém menos que q coeficientes não nulos), e cada coluna representa um patch $\boldsymbol{\gamma_i}$. A matriz $\boldsymbol{V}$ está organizada de modo que o coeficiente de maior magnitude esteja sempre nas linhas de menor índice. Assim, para uma coluna j, se na linha i for encontrado um valor 0 (i.e., coeficiente não necessário), significa que as demais linhas da coluna também serão 0. Logo, ao encontrar um valor 0 numa coluna, apenas este precisa ser inserido no vetor $\boldsymbol{r}$, pois se sabe que o próximo índice já será relativo a próxima coluna. Dessa forma, os vetores $\boldsymbol{v}$ e $\boldsymbol{r}$ serão os seguintes:

$$\boldsymbol{v} = [189, 45, 8, 36, 114, 32],$$
$$\boldsymbol{r} = [7{,}1{,}5{,}12{,}0{,}15{,}3{,}0].$$

Ainda em tempo, considerando que alguns dos patches não foram recalculados, estes também não precisam ser reenviados. Para marcar em que ponto isso acontece foi

utilizado o valor 254 em $\boldsymbol{r}$, seguido do número de patches não enviados.

Esses vetores, mais a informação de máximo, mínimo e q precisam ser empacotados e comprimidos para enviar ao decoder. A compressão utilizada foi baseada em codificação de Huffman, e foi utilizada a biblioteca *Boost Gzip* (BOOST, 2018) para tal. O Apêndice A descreve o processo de codificação de Huffman.

4.6 Descompressão e dequantização

A primeira etapa do processo de decodificação consiste na descompressão dos dados recebidos. É executada a decodificação de Huffman, baseando-se na tabela de compressão gerada para recuperar os valores originais. A biblioteca *Boost Gzip* (BOOST, 2018) foi utilizada novamente com esse objetivo. A partir do sinal descomprimido deve-se recuperar as matrizes $\boldsymbol{V}$ e $\boldsymbol{R}$ tomando os vetores $\boldsymbol{v}$ e $\boldsymbol{r}$ recebidos. Em seguida é necessário recompor os coeficientes quantizados executando a operação inversa a de quantização. Considerando c' cada coeficiente recebido, u o maior valor e l o menor, valores todos recebidos no *bitstream*,

$$c = \frac{c'(u-l)}{255} + l.$$

Por fim, é inicializada com zeros uma matriz $\boldsymbol{\Gamma} \in \mathbb{R}^{K \times T}$. Em seguida, associa-se a cada índice $R_{i,j}$ de $\boldsymbol{\Gamma}$ o coeficiente $V_{i,j}$ correspondente, para todo $1 \leq i \leq T$ e $1 \leq j \leq q$. Vale lembrar que alguns dos patches podem não ter sido recebidos. Essa informação deve ser passada adiante para que somente os patches modificados sejam reprocessados. Terminada essa etapa procede-se com a recuperação dos patches.

4.7 Recuperação dos patches

A recuperação dos patches por parte do *decoder* é um processo muito simples, pois incorre somente em uma multiplicação entre o dicionário $\boldsymbol{D}$ e a representação esparsa recebida $\boldsymbol{\Gamma}$. Assim, pode-se definir que a matriz de patches $\boldsymbol{X} = \boldsymbol{D\Gamma}$. Uma simplificação que pode aumentar a velocidade da decodificação é não criar a matriz $\boldsymbol{\Gamma}$, ou seja, utilizar as informações de $\boldsymbol{V}$ e $\boldsymbol{R}$ para formar $\boldsymbol{X}$. Esse procedimento tende a ser mais eficiente, pois quando formada a matriz $\boldsymbol{\Gamma}$, a maior parte dos coeficientes dessa matriz é nulo, por se tratar de uma representação esparsa.

4.8 Reconstrução do quadro

A reconstrução do quadro se dá exatamente da forma contrária a separação por patches. Vale notar que alguns dos patches não precisarão ser atualizados (i. e., não se modificaram de um quadro para outro). Essa informação foi recebida pelo *decoder* na Seção 4.6 e repassada até essa etapa.

Assim, cada um dos patches de $\boldsymbol{X}$, que estavam vetorizados, são novamente transformados em matrizes de dimensão $p \times p \times 3$, e colocados em sua devida posição, formando a imagem recebida $\boldsymbol{I}$ de dimensões $m \times n$.

4.9 Implementação

A implementação do codec se deu utilizando a linguagem de programação Python para a etapa de aprendizado de dicionários, e, como já citado, a biblioteca SPAMS (MAIRAL, 2009). Para o *encoder* e *decoder* foi escolhida a linguagem C++. O carregamento de vídeos utilizou a biblioteca OpenCV (OPENCV, 2018). Já para o processamento das matrizes na memória da CPU, a biblioteca Eigen (EIGEN, 2018) foi a selecionada.

O programa para testes executa os processos de codificação e decodificação, realizando métricas de tempo e qualidade para ambas etapas. Além disso, a parte de codificação salva um arquivo comprimido com o resultado da mesma. Um programa que somente desempacota os dados e decodifica também foi desenvolvido.

Além disso, foram criados alguns subprogramas com o intuito de facilitar os testes e execuções, bem como analisar outras possibilidades, e executar métricas em paralelo ao desenvolvimento do codec.

5 RESULTADOS

De modo a avaliar o desempenho da abordagem proposta, foram realizados experimentos com vídeos utilizados na literatura para testes, disponíveis em <https://media.xiph.org/video/derf/>. Os vídeos estão em formato de sequência de quadros, sem compressão, ou somente com compressão sem perdas. Alguns dos vídeos disponíveis foram escolhidos. Estes foram classificados nos seguintes grupos: *cartoon*, monitoramento, esportes, *indoor* e natureza, de forma que fosse possível verificar se existe alguma diferença no uso da técnica para diferentes classes de vídeos. A tabela a seguir ilustra as escolhas. Os vídeos em negrito foram os escolhidos para a validação, enquanto os demais para treinamento.

Tabela 5.1: Vídeos utilizados para treinamento e validação (mostrados em negrito).

Categoria	*Vídeo*	*Resolução*	*Nº de quadros*
Cartoon	**sintel_trailer**	1024×436	1.253
	big_buck_bunny	720p	14.315
	elephants_dream	1024×436	15.691
Esportes	**football**	NTSC	360
	stefan	SIF	300
	tennis	SIF	150
Indoor	**johnny**	720p	600
	carphone	QCIF	382
	deadline	CIF	1.374
	foreman	CIF	300
	hall_monitor	CIF	300
	mother_daughter	QCIF	961
Monitoramento	**stockholm**	720p	604
	bridge_close	CIF	2.001
	bridge_far	CIF	2.101
	station	1080p	313
Natureza	**park_joy**	720p	500
	ducks_take_off	720p	500
	garden	SIF	115
	in_to_tree	720p	500
	park_run	720p	504
	waterfall	CIF	260

Fonte: O Autor.

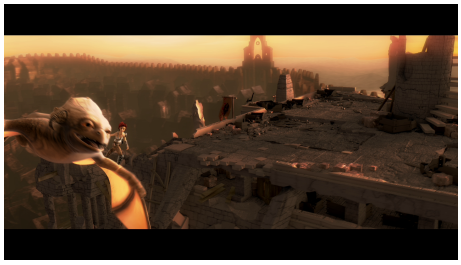
A máquina utilizada para treinamento e testes possui um processador Intel i5 2.5 GHz e 4 cores físicos, e 8Gb de memória RAM. Além disso, a placa de vídeo é Nvidia Geforce GTX 1050Ti, com 4Gb de memória.

Para ilustrar os vídeos utilizados na avaliação de resultados, a figura abaixo apresenta uma imagem de cada vídeo, para que se possa perceber as peculiaridades inerentes

a cada um dos vídeos escolhidos.

Figura 5.1: Imagens dos vídeos escolhidos para os testes. (a) sintel_trailer. (b) football. (c) johnny. (d) stockholm. (e) park_joy.

(a)



(b)

(c)

(d)

(e)

Fonte: O Autor.

Em seguida serão discutidos o treinamento e os testes realizados com os vídeos.

5.1 Treinamento

Foi gerado um dicionário para cada classe de vídeos, e para cada variação em K (tamanho do dicionário) e p (tamanho do patch), utilizando o algoritmo ODL, da forma que foi descrito na proposta do trabalho.

Para o treinamento foram utilizados os vídeos de cada classe, com exceção daquele que foi escolhido para teste. De cada um dos conjuntos de vídeos foram amostrados quadros a cada t segundos, variando o valor de t para providenciar pelo menos 10 e no máximo 100 quadros para o processo de treinamento, por classe de vídeos. A ordem de processamento dos quadros foi aleatória, para evitar viés para os últimos quadros. Cada quadro foi então transformado em sequência de patches, com sobreposição. Essa

sequência foi introduzida no algoritmo ODL, que tomou no máximo 2.500 amostras por sequência para fazer o treinamento do dicionário.

A tabela a seguir apresenta os parâmetros de cada dicionário criado para os testes subsequentes. Cada parametrização foi aplicada a todas as classes de vídeos. É importante recordar que o parâmetro p trata do tamanho do patch $p \times p \times 3$, enquanto K representa o número de átomos do dicionário.

Tabela 5.2: Parâmetros tamanho do patch p e tamanho do dicionário K utilizados para treinamento e validação.

Tamanho do patch	4	4	4	4	4	8	8	8	8
Tamanho do dicionário	16	24	32	48	64	32	64	128	192

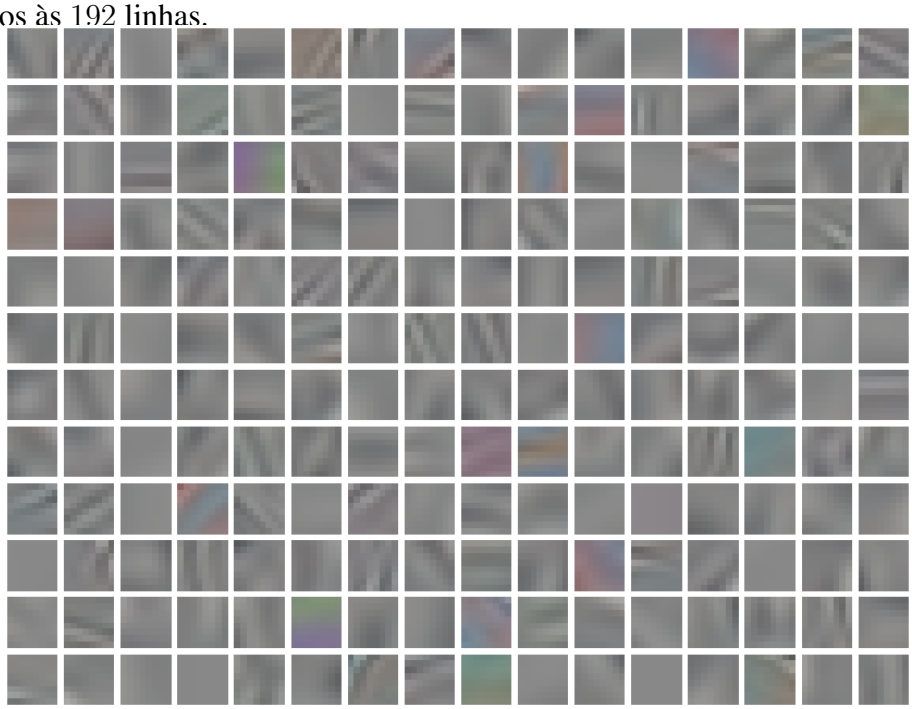
Fonte: O Autor.

Os parâmetros para teste foram escolhidos com base na implementação, também considerando o padrão H.264, que utiliza, na etapa de transformação, patches de 4×4, e o padrão JPEG, cujos patches são 8×8 (MITCHELL, 1992). O valor de s, parâmetro de esparsidade (Tabela 4.1), foi fixo para os valores de p. Sendo assim, para $p = 4$, $s = 4$, e para $p = 8$, $s = 8$.

A Figura 5.2 mostra um dicionário aprendido para a classe cartoon com patches 8×8. Isto resulta em um dicionário contendo 192 linhas ($8 \times 8 \times 3$). Neste exemplo, o dicionário também contém 192 colunas. Cada quadrado mostrado contém 8×8 pixels, sendo cada pixel exibindo um valore RGB. O conteúdo de cada um dos quadrados corresponde a uma coluna, com os valores associados às 192 linhas.

A Figura 5.3 mostra um outro exemplo de dicionário aprendido para a classe indoors com patches de 4×4. Neste exemplo, o dicionário contém 48 linhas ($4 \times 4 \times 3$) e 32 colunas (número de quadrados mostrados na figura).

Figura 5.2: Exemplo de dicionário aprendido para a classe cartoon com patches 8×8. Isto resulta em um dicionário contendo 192 linhas ($8 \times 8 \times 3$). Este dicionário contém 192 colunas. Cada quadrado mostrado contém 8×8 pixels, cada um exibindo um valore RGB. O conteúdo de cada um dos quadrados corresponde a uma coluna, com os valores associados às 192 linhas.



Fonte: O Autor.

Figura 5.3: Exemplo de dicionário aprendido para a classe cartoon com patches 4×4. Isto resulta em um dicionário contendo 48 linhas ($4 \times 4 \times 3$). Este dicionário contém 32 colunas. Cada quadrado mostrado contém 4×4 pixels, cada um exibindo um valore RGB. O conteúdo de cada um dos quadrados corresponde a uma coluna, com os valores associados às 48 linhas.

Fonte: O Autor.

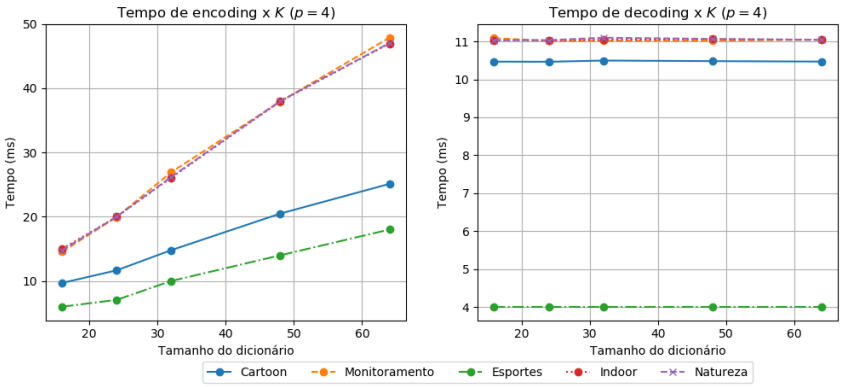
Em seguida serão apresentados os resultados alcançados com as diferentes parametrizações e diferentes classes de vídeos.

5.2 Alterações no tamanho do dicionário, no tamanho do patch e no parâmetro de esparsidade

Nessa seção serão apresentados e discutidos resultados alcançados na compressão e decompressão dos vídeos escolhidos de cada classe para os parâmetros definidos na Tabela 5.2. No total foram 9 parametrizações diferentes para cada uma das 5 classes de vídeos, totalizando 45 combinações diferentes para serem testadas.

Ao observar o algoritmo OMP (1), verifica-se que sua complexidade teórica é $O(smd)$, tal que, nesse caso, $m = K$ e $d = 3p^2$, portanto, $O(sKp^2)$. Os resultados mostram o que a teoria espera, um aumento linear no tempo de processamento, pois no caso considerado, foram mantidos s e d fixos, enquanto a variação ocorreu em m. Vale ressaltar que o processamento de cada patch foi individual e utilizando a plataforma CUDA®, que paraleliza as operações. Porém, por mais que patches possam ser executados em paralelo, seu tempo individualmente, no processamento do OMP, aumenta conforme o tamanho do dicionário.

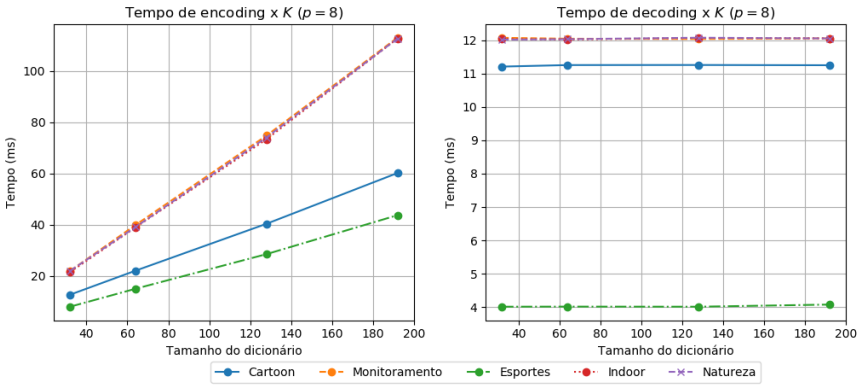
Figura 5.4: Tempo médio para codificação e decodificação por quadro para patches 4×4 em função do tamanho do dicionário K.



Fonte: O Autor.

Na figura acima (5.4) também pode-se constatar que a decodificação tem tempos iguais, independentes do tamanho do dicionário. Isso se dá pelo fato de que não é feita a multiplicação pelo método formal, mas sim tomando somente cada um dos s valores não nulos do vetor. O mesmo pode ser constatado na figura abaixo, que apresenta os resultados alcançados com $p = 8$.

Figura 5.5: Tempo médio para codificação e decodificação por quadro para patches 8×8 em função do tamanho do dicionário K.



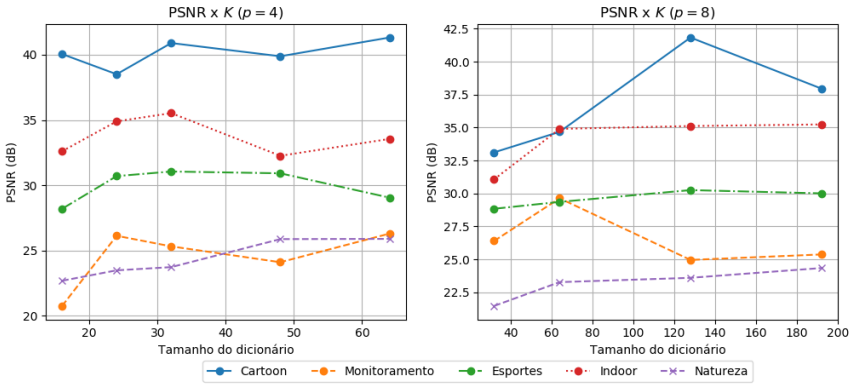
Fonte: O Autor.

Um fato importante a ser detalhado é de que a classe esportes teve menor tempo médio por quadro. Esse fato está simplesmente ligado a resolução do vídeo utilizado por essa classe, NTSC, que é aproximadamente $2{,}63$ vezes menor que 720p. O mesmo acontece com a classe cartoon, no qual o vídeo escolhido possui resolução menor e, portanto, menor tempo de codificação. Percebe-se, portanto, uma relação linear entre o tamanho do quadro e o tempo de codificação.

Além disso, o vídeo de cartoon possui vários quadros com partes que se repetem. O sistema simples que não recalcula os patches que são iguais entrou em ação, fazendo com que o tempo médio para decodificação fosse ainda um pouco menor.

Também foi verificada a qualidade dos vídeos utilizando o PSNR médio. Os resultados mostram que o PSNR não se alterou significativamente para os diversos tamanhos de dicionário, o que indica que é possível utilizar valores baixos de K sem perdas significativas na qualidade da representação.

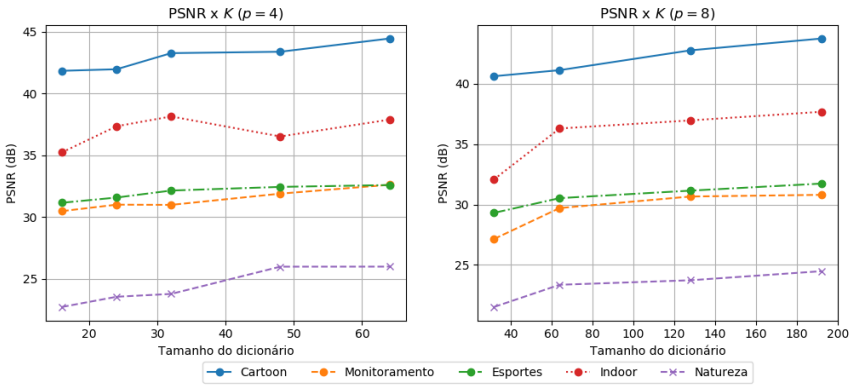
Figura 5.6: PSNR médio em função do tamanho do dicionário K, com quantização de 8 bits.



Fonte: O Autor.

Nos gráficos da Figura 5.6 verifica-se uma variabilidade no PSNR médio devido a quantização. Para melhorar esse fato, poderiam ser utilizados inteiros de 16 bits ao invés de 8 bits na representação comprimida, considerando que, neste caso, a representação perde significativamente em compressibilidade. A Figura 5.7 mostra que o PSNR médio é bem mais estável utilizando uma representação de 16 bits para todas as classes.

Figura 5.7: PSNR médio em função do tamanho do dicionário K, com quantização de 16 bits.



Fonte: O Autor.

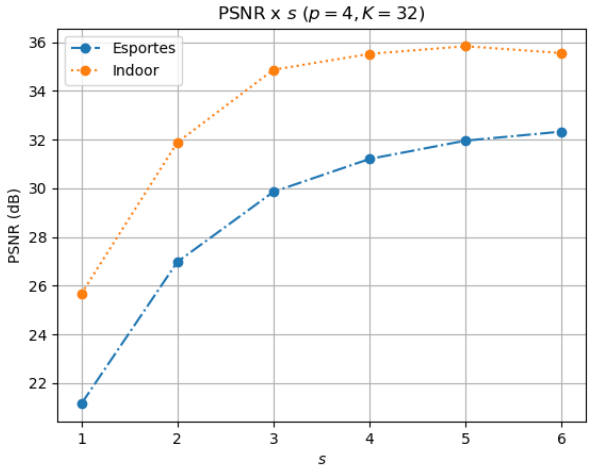
Além de alcançar maior estabilidade utilizando 16 bits na quantização, também é possível verificar que acontece alteração no PSNR devido ao uso dos 16 bits, podendo ser uma alternativa para melhoria na representação.

Pode-se verificar, ainda, com o uso de 16 bits, que o PSNR aumenta em função do tamanho do dicionário, de uma forma geral. Uma exceção está nos vídeos indoor, que,

para tamanho do patch 4 e tamanho do dicionário 48 tem um resultado de PSNR médio um pouco pior. Porém, apesar da diferença de PSNR confrome o tamanho do dicionário, essa não é tão grande, o que mostra que o uso de dicionários menores pode ser bastante útil, especialmente quando se leva em conta a velocidade de codificação.

Em seguida, foi escolhida a melhor representação do vídeo indoor, que apresenta pouca variação nas cenas, para avaliar as mudanças no PSNR médio em função da esparsidade. Também foi escolhido o vídeo de esportes, usando a mesma configuração, este pelo fato de que possui patches com grama, que é uma imagem com alta frequência. Sumarizando, foram escolhidos $p = 4$, $K = 32$, e $s = 1,2,3,4,5,6$.

Figura 5.8: PSNR médio em função do parâmetro de esparsidade s, para patches 4×4 ($p = 4$) e tamanho do dicionário $K = 32$.



Fonte: O Autor.

O que se pode perceber é que conforme o valor de s aumenta, o PSNR médio também, mas não de forma linear. Os pontos $s = 4,5,6$ de ambos vídeos não apresentam tão importantes melhoras no PSNR médio na relação entre eles. Inclusive, para $s = 6$, o vídeo indoor apresentou degradação em relação a $s = 5$, causada pelos erros de quantização envolvidos. Dadas essas constatações, pode-se verificar que o aumento do valor de s faz com que a representação seja capaz de aprimorar as altas frequências, muito semelhante ao uso da DCT.

Por fim, foi aplicada a métrica de *bitrate*, que consiste em calcular quantos bits por segundo a representação compactada de cada vídeo conseguiu alcançar. A partir da codificação salva dos vídeos codificados, foi percebido que as melhores compressões aconteceram com tamanho de patch $p = 8$, variando pouco em relação ao tamanho do dicionário. Além disso, é possível verificar que o PSNR médio não apresenta mudança

significativa em relação à mudança do tamanho do patch ou do dicionário, como já citado anteriormente, o que proporciona a possibilidade de se escolher a representação mais compacta para comparar com o codec H.264. Portanto, foi calculado o bitrate usando o tamanho do patch em 8 e o tamanho do dicionário em 128.

Tabela 5.3: Bitrate calculado para vídeos de teste com tamanho do patch 8, tamanho do dicionário 128 e parâmetro de esparsidade 8.

Classe	*Vídeo*	*Nº quadros*	*FPS*	*Tam. arquivo (bytes)*	*Bitrate (kbit/s)*
Cartoon	sintel_trailer	1.253	24	79.674.062	11.922,5
Esportes	football	360	30	21.420.945	13.945,9
Indoor	johnny	600	60	72.782.464	56.861,3
Monitor.	stockholm	604	59,94	82.818.027	64.208,8
Natureza	park_joy	500	50	80.040.734	62.531,8

Fonte: O Autor.

A informação de quantos bytes a representação possui foi verificada através do salvamento de um arquivo por teste com a codificação apresentada na proposta do trabalho.

É possível observar que, para vídeos com a câmera em movimento, caso dos dois últimos vídeos da Tabela 5.3, o valor do o bitrate degrada consideravelmente, chamando a atenção da necessidade de se agregar possivelmente algum tipo de detecção de movimento. Em especial é possível notar que o maior vídeo (sintel_trailer) teve maior compressão. Além de ser o maior vídeo, o mesmo também possui diversas partes com a câmera parada, o que facilita a compressão devido ao método proposto.

Fica claro que, quando se deseja tempo real na codificação, deve-se utilizar valores de tamanho de patch, parâmetro de esparsidade e tamanho de dicionário mais baixos, enquanto para que haja uma representação mais comprimida, devem ser utilizados valores mais altos. O parâmetro de esparsidade s é capaz de controlar a qualidade, a compressão e até mesmo a velocidade da codificação, sendo então necessário considerar várias possibilidades para seus valores.

Para resumir, a Tabela 5.4 traz recomendações para os valores dos parâmetros do codec, nos casos em que o foco está em tempo real ou compressão.

Tabela 5.4: Valores recomendados de parâmetros em função da aplicação desejada.

Aplicação	*Foco*	*Tam. patch*	*Tam. dicionário*	*Par. de esparsidade*
Tempo real	qualidade	4	16	4
Tempo real	compressão	8	64	5
Gravação	qualidade	8	128	10
Gravação	qual. + compr.	8	128	8
Gravação	compressão	8	128	4

Fonte: O Autor.

5.3 Comparação com codec H.264

Para a comparação com o codec H.264 foi utilizado o software FFmpeg (FFMPEG, 2018), tendo como entrada as mesmas sequências que serviram para os testes executados. O software permite que se gere arquivos de saída com aproximadamente um valor de bitrate desejado. Dessa forma, foi executado o programa para alcançar os mesmos bitrates que os apresentados na Tabela 5.3. O codec H.264 permite diversas parametrizações. Por questão de simplicidade, foi utilizada a configuração padrão, apenas forçando o bitrate desejado. Em seguida, foi calculado o PSNR médio dos vídeos gerados com o codec H.264.

Tabela 5.5: Comparação PSNR médio para mesmo bitrate.

Classe	*Vídeo*	*Bitrate (kbit/s)*	*PSNR Proposta (dB)*	*PSNR H.264 (dB)*
Cartoon	sintel_trailer	11.922,5	41,83	54,96
Esportes	football	13.945,9	24,96	36,55
Indoor	johnny	56.861,3	30,25	37,50
Monitor.	stockholm	64.208,8	35,11	42,96
Natureza	park_joy	62.531,8	23,60	32,37

Fonte: O Autor.

Os resultados obtidos tiveram, para todos os casos, valores de PSNR médio inferiores aos obtidos com o uso do codec H.264, o que já era esperado. A abordagem proposta é bastante simples, enquanto que o padrão H.264 é amplamente desenvolvido e foi aprimorado tanto pela academia quanto pela indústria. Porém, é possível perceber que o codec proposto funciona, porém precisa ser aprimorado. Em especial, buscar utilizar as técnicas já conhecidas dos padrões de codec, por exemplo predição de movimento, pode ser uma via para melhorias no PSNR médio e no bitrate.

Para possibilitar uma melhor visualização da situação, também foi executado o software FFmpeg codificando os vídeos para H.264 sem utilizar nenhuma parametrização, ou seja, utilizando o padrão do programa. Os resultados são mostrados na Tabela 5.6.

Tabela 5.6: PSNR médio e bitrate usando codificação padrão.

Classe	*Vídeo*	*Bitrate (kbit/s)*	*PSNR H.264 (dB)*
Cartoon	sintel_trailer	901,15	45,58
Esportes	football	2.624,67	32,85
Indoor	johnny	874,84	37,93
Monitoramento	stockholm	3.703,45	31,96
Natureza	park_joy	14.828,28	27,26

Fonte: O Autor.

Os experimentos realizados indicam que a opção pelo uso das técnicas propostas,

agregados a melhorias que possam ser executadas, podem gerar resultados ainda melhores.

5.4 Resultado visual

Por fim, serão observados alguns resultados visuais relativos ao codec proposto. Os resultados discutidos apresentam erros nos quadros gerados pelo codec proposto, comparando a imagem de referência com a gerada pela codificação.

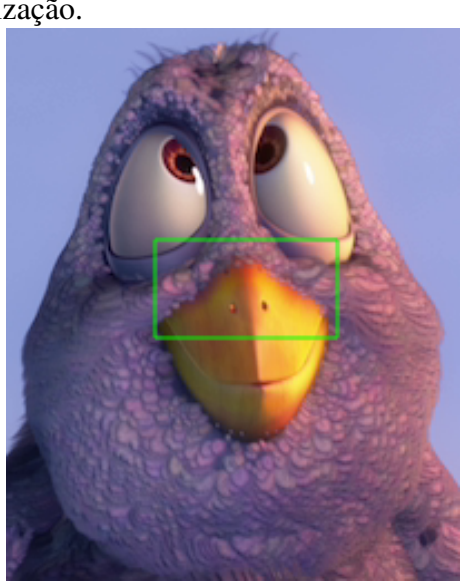
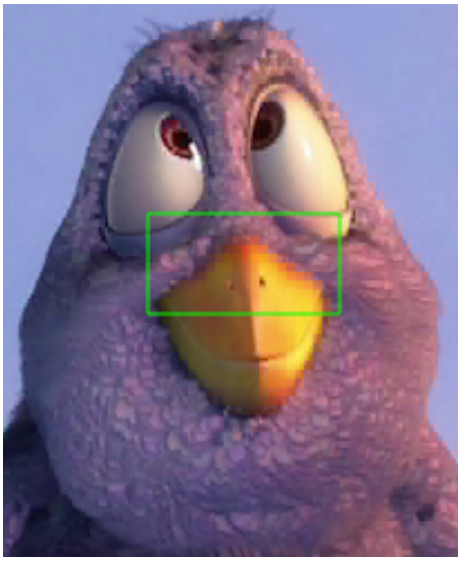
Figura 5.9: Fundo de cena do vídeo johnny, tamanho do dicionário $K = 32$, parâmetro de esparsidade $s = 5$ e patches 4×4, quantizado 8 bits. (a) Referência. (b) Resultado com codec proposto.

(a)  (b) 

Fonte: O Autor.

Na Figura 5.9 pode-se verificar que altas frequências foram adicionadas a imagem, na comparação entre a imagem de referência e a imagem gerada pelo codec.

Figura 5.10: Vídeo big_buck_bunny, tamanho do dicionário $K = 16$, parâmetro de esparsidade $s = 5$ e patches 4×4, quantizado 8 bits. (a) Referência. (b) Resultado com codec proposto. Um retângulo delimita uma região onde é possível perceber artefatos de quantização.

(a)  (b) 

Fonte: O Autor.

Já nesse caso, o quadro resultado é bem próximo ao original. O que se nota, apesar disso, é que em alguns pontos da imagem foram criados artefatos relacionados ao fato da imagem ter sido dividida em patches. Para reduzir um pouco os efeitos, mas sem causar ofuscamento na imagem, foi utilizado um filtro bilateral. Esta não é a solução ideal, dado que existem algoritmos para especificamente filtrar esses blocos, porém na versão final do codec não foi implementada essa funcionalidade.

6 CONCLUSÃO

O trabalho apresentou uma proposta de codificação e decodificação de vídeo em tempo real utilizando métodos de *compressed sensing* aliados a aprendizado de dicionários. O desempenho do codec proposto foi avaliado considerando métricas como PSNR e bitrate em comparação com o padrão H.264, amplamente utilizado para codificação de vídeos.

Na fase de execução de testes e geração de resultados foi possível constatar o funcionamento do codec para 5 tipos diferentes de classes de vídeos. Ficou clara também a relação entre execução em tempo real, compressibilidade do vídeo e qualidade com os parâmetros K (número de átomos do dicionário), p (tamanho do bloco), e s (parâmetro de esparsidade da representação).

Mudanças no valor de K (tamanho do dicionário) acarretam pouca diferença na qualidade do vídeo, desde que o valor não seja muito pequeno. Por outro lado, o valor de K altera linearmente o tempo de codificação dos quadros, como pode ser verificado nos gráficos das Figuras 5.4 e 5.5. Também se percebe que o tempo de decodificação não se altera. A compressibilidade é alterada pelo valor de K, sendo que valores maiores correspondem a maior capacidade de compressão (i.e., resultam em arquivos de menor tamanho). Resumindo, se o objetivo for maior compressão, deve-se escolher valores maiores de K. Se a ideia é tempo real, valores menores são necessários. Vale lembrar que os dicionários precisam ser previamente gerados. O que pode valer a pena, nesse caso, é escolher, para determinada classe, um dicionário para tempo real e outro para obter maior compressão, através de testes.

Quanto ao tamanho do patch p, esse produz dois distúrbios no codec. Em primeiro lugar, ao observar $p = 4$ e $p = 8$, fica claro que, para tempo real, $p = 4$ é uma melhor solução. Para maior compressão, $p = 8$. Seria interessante serem executados testes com valores de p maiores que 8, no sentido de verificar a capacidade de compressão. Para isso, em primeiro lugar, deve-se trabalhar em tornar mais veloz a codificação, tanto melhorando a implementação do algoritmo utilizado, ou buscando outras soluções. Por exemplo, durante a execução do trabalho, não foi possível verificar o funcionamento paralelo do algoritmo IHT (BLUMENSATH; DAVIES, 2009), que claramente apresenta características interessantes para um algoritmo paralelizável. Um detalhe importante é que o tamanho do patch não altera significativamente a qualidade da representação, podendo ser alcançada boa qualidade com tamanhos diferentes. Porém, vale ressaltar que o

uso de patches sem sobreposição tende a produzir artefatos nos quadros de vídeos comprimidos. Para minimizar este problema foram criados algoritmos para *deblocking*, que poderiam ser aplicados ao codec proposto.

Já no que diz respeito ao parâmetro de esparsidade s, ficou claro que ele altera a qualidade da representação e também a compressibilidade. Quanto à compressibilidade, o resultado pode ser verificado pela própria representação dos dados. Considere um quadro 720p usando $p = 4$, utilizando 1 byte para o valor e 1 byte para o índice em cada patch, e que são 57600 patches, $112{,}5$ kbytes serão utilizados aproximadamente a cada incremento em s.

Por fim, as comparações com o codec H.264 mostraram que existem muitos pontos que podem ser aprimorados, mas também mostra que a proposta apresentada pode ser construtiva. Em relação aos padrões já existentes, fica clara a necessidade de maior desenvolvimento em todas as dimensões do codec, pois o que foi apresentado ilustra a ideia principal. Um caminho a ser seguido é combinar as metodologias já existentes para predição intra-quadro e inter-quadro com o aprendizado de dicionários, além de estruturar essa relação. Além disso, é necessário verificar formas de implementar a camada de rede (i.e. parte do codec que define como transmissor e receptor irão se comunicar) do codec baseando-se nos padrões já existentes.

De forma resumida, este trabalho apresentou uma proposta para a codificação de vídeos utilizando aprendizado de dicionários e técnicas de compressed sensing. Do ponto de vista do aprendizado, o trabalho proposto permitiu que, através da pesquisa e do estudo, fosse possível compreender e dominar melhor os assuntos base da proposta, que não são estudados no escopo das disciplinas de graduação.

REFERÊNCIAS

BLUMENSATH, T.; DAVIES, M. E. Iterative hard thresholding for compressed sensing. **Applied and Computational Harmonic Analysis**, v. 27, n. 3, p. 265 – 274, 2009. ISSN 1063-5203. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1063520309000384>.

BOOST. **Gzip Filters**. 2018. Available from Internet: <https://www.boost.org/doc/libs/1_68_0/libs/iostreams/doc/classes/gzip.html>.

BRYT, O.; ELAD, M. Compression of facial images using the k-svd algorithm. **Journal of Visual Communication and Image Representation**, v. 19, n. 4, p. 270 – 282, 2008. ISSN 1047-3203. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1047320308000254>.

CANDES, E. J. et al. Compressed sensing with coherent and redundant dictionaries. **Applied and Computational Harmonic Analysis**, v. 31, n. 1, p. 59 – 73, 2011. ISSN 1063-5203. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1063520310001156>.

CANDES, E. J.; ROMBERG, J.; TAO, T. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. **IEEE Transactions on Information Theory**, v. 52, n. 2, p. 489–509, Feb 2006. ISSN 0018-9448.

CANDES, E. J.; ROMBERG, J. K.; TAO, T. Stable signal recovery from incomplete and inaccurate measurements. **Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences**, Wiley Online Library, v. 59, n. 8, p. 1207–1223, 2006.

CANDES, E. J.; TAO, T. Decoding by linear programming. **IEEE Transactions on Information Theory**, v. 51, n. 12, p. 4203–4215, Dec 2005. ISSN 0018-9448.

CANDES, E. J.; TAO, T. Near-optimal signal recovery from random projections: Universal encoding strategies? **IEEE Transactions on Information Theory**, v. 52, n. 12, p. 5406–5425, Dec 2006. ISSN 0018-9448.

CHEN, G.; NEEDELL, D. Compressed sensing and dictionary learning. **Preprint**, v. 106, 2015.

CHEN, H.-W.; KANG, L.-W.; LU, C.-S. Dictionary learning-based distributed compressive video sensing. In: CITESEER. **Picture Coding Symposium (PCS), 2010**. [S.l.], 2010. p. 210–213.

DO, T. T. et al. Distributed compressed video sensing. In: **2009 43rd Annual Conference on Information Sciences and Systems**. [S.l.: s.n.], 2009. p. 1–2.

DONOHO, D. L. Compressed sensing. **IEEE Transactions on Information Theory**, v. 52, n. 4, p. 1289–1306, April 2006. ISSN 0018-9448.

EIGEN. **Eigen Overview**. 2018. Available from Internet: <http://eigen.tuxfamily.org/index.php?title=Main_Page>.

ELAD, M.; AHARON, M. Image denoising via sparse and redundant representations over learned dictionaries. **IEEE Transactions on Image Processing**, v. 15, n. 12, p. 3736–3745, Dec 2006. ISSN 1057-7149.

FFMPEG. **FFmpeg**. 2018. Available from Internet: <https://www.ffmpeg.org/>.

HAYKIN, S. S.; VEEN, B. V. **Sinais e sistemas**. [S.l.]: Bookman, 2001.

HUFFMAN, D. A. A method for the construction of minimum-redundancy codes. **Proceedings of the IRE**, v. 40, n. 9, p. 1098–1101, Sept 1952. ISSN 0096-8390.

LIMA, V. d. et al. Codificação de vídeo baseada em fractais e representações esparsas. [sn], 2012.

MAIRAL, J. **SPArse Modeling Software**. 2009. Available from Internet: <http://spams-devel.gforge.inria.fr/index.html>.

MAIRAL, J.; BACH, F.; PONCE, J. Sparse modeling for image and vision processing. **Found. Trends. Comput. Graph. Vis.**, Now Publishers Inc., Hanover, MA, USA, v. 8, n. 2-3, p. 85–283, dec. 2014. ISSN 1572-2740. Available from Internet: <http://dx.doi.org/10.1561/0600000058>.

MAIRAL, J. et al. Online dictionary learning for sparse coding. In: **Proceedings of the 26th Annual International Conference on Machine Learning**. New York, NY, USA: ACM, 2009. (ICML '09), p. 689–696. ISBN 978-1-60558-516-1. Available from Internet: <http://doi.acm.org/10.1145/1553374.1553463>.

MCCUE, T. **Video Marketing In 2018 Continues To Explode As Way To Reach Customers**. 2018. Available from Internet: <https://www.forbes.com/sites/tjmccue/2018/06/22/video-marketing-2018-trends-continues-to-explode-as-the-way-to-reach-customers/#9d77d9e598dc>.

MENDELSON, S.; PAJOR, A.; TOMCZAK-JAEGERMANN, N. Uniform uncertainty principle for bernoulli and subgaussian ensembles. **Constructive Approximation**, v. 28, n. 3, p. 277–289, Dec 2008. ISSN 1432-0940. Available from Internet: <https://doi.org/10.1007/s00365-007-9005-8>.

MITCHELL, J. Digital compression and coding of continuous-tone still images: Requirements and guidelines. **ITU-T Recommendation T**, v. 81, 1992.

MUTHUKRISHNAN, S. Data streams: Algorithms and applications. **Foundations and Trends® in Theoretical Computer Science**, v. 1, n. 2, p. 117–236, 2005. ISSN 1551-305X. Available from Internet: <http://dx.doi.org/10.1561/0400000002>.

NEEDELL, D.; TROPP, J. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. **Applied and Computational Harmonic Analysis**, v. 26, n. 3, p. 301 – 321, 2009. ISSN 1063-5203. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1063520308000638>.

NVIDIA. **CUDA C Programming Guide**. 2018. Available from Internet: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>.

NVIDIA. **CUDA Zone**. 2018. Available from Internet: <https://developer.nvidia.com/cuda-zone>.

NVIDIA. **Dense Linear Algebra on GPUs**. 2018. Available from Internet: <https://developer.nvidia.com/cublas>.

NYQUIST, H. Certain topics in telegraph transmission theory. **Transactions of the American Institute of Electrical Engineers**, v. 47, n. 2, p. 617–644, April 1928. ISSN 0096-3860.

OPENCV. **OpenCV (Open Source Computer Vision Library)**. 2018. Available from Internet: <https://opencv.org/>.

PRADES-NEBOT, J.; MA, Y.; HUANG, T. Distributed video coding using compressive sampling. In: **2009 Picture Coding Symposium**. [S.l.: s.n.], 2009. p. 1–4.

RUDELSON, M.; VERSHYNIN, R. On sparse reconstruction from fourier and gaussian measurements. **Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences**, Wiley Online Library, v. 61, n. 8, p. 1025–1045, 2008.

SULLIVAN, G. J.; WIEGAND, T. Video compression - from concepts to the h.264/avc standard. **Proceedings of the IEEE**, v. 93, n. 1, p. 18–31, Jan 2005. ISSN 0018-9219.

TROPP, J. A.; GILBERT, A. C. Signal recovery from random measurements via orthogonal matching pursuit. **IEEE Transactions on Information Theory**, v. 53, n. 12, p. 4655–4666, Dec 2007. ISSN 0018-9448.

WIEGAND, T. et al. Overview of the h.264/avc video coding standard. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 13, n. 7, p. 560–576, July 2003. ISSN 1051-8215.

XIANG, S.; CAI, L. Scalable video coding with compressive sensing for wireless videocast. In: **2011 IEEE International Conference on Communications (ICC)**. [S.l.: s.n.], 2011. p. 1–5. ISSN 1938-1883.

ZHANG, Y. et al. A novel image/video coding method based on compressed sensing theory. In: **2008 IEEE International Conference on Acoustics, Speech and Signal Processing**. [S.l.: s.n.], 2008. p. 1361–1364. ISSN 1520-6149.

APÊNDICES

APÊNDICE A
Codificação de Huffman

Dentre os diversos tipos de codificação de dados que existem, alguns deles permitem a representação compacta dos dados sem perdas de compressão. Esse é o caso da codificação de Huffman, um algoritmo de compressão por entropia, ou seja, que se aproveita da repetibilidade dos dados e de sua distribuição, pois, em geral, sinais que carregam informação possuem bastante redundância.

Considere que deseja-se comprimir uma sequência de símbolos, ou mensagem, utilizando códigos de tamanho variável, um para cada símbolo. Segundo Huffman (1952), duas restrições básicas são impostas para a codificação de mensagens proposta pelo autor:

1. Dois símbolos diferentes não podem ter o mesmo código;
2. Os códigos das mensagem precisam ser construídos de forma que não seja necessário identificar onde começa ou termina um código.

Um princípio básico utilizado é que o tamanho de um código nunca pode ser menor do que o de outro menos provável (HUFFMAN, 1952).

Para facilitar as explicações, dados N símbolos, eles serão representados pelos numerais $1, 2, ..., N$. Algo que pode ser assumido, sem perda de generalidade, considerando $P(s)$ a probabilidade de um símbolo a qualquer aparecer na mensagem, é que (HUFFMAN, 1952):

$$P(1) \geq P(2) \geq \cdots \geq P(N). \tag{6.1}$$

Por consequência, se quisermos uma codificação ótima, o tamanho do código $L(s)$ deve ser (HUFFMAN, 1952):

$$L(1) \leq L(2) \leq \cdots \leq L(N). \tag{6.2}$$

Pode-se definir, a partir de (6.1) e (6.2) o tamanho L da mensagem codificada:

$$L = \sum_{i=1}^{N} P(i)L(i).$$

É importante ressaltar que:

$$\sum_{i=1}^{N} P(i) = 1.$$

Por fim, Huffman (1952) apresentou três novas restrições, além das iniciais. Considere-se D o número de valores que um dígito do código pode assumir.

1. $L(1) \leq L(2) \leq \cdots \leq L(N-1) = L(N)$.
2. Pelo menos 2 e não mais que D símbolos com código de tamanho $L(N)$ tem códigos com início idêntico, diferenciando-se por seus dígitos finais.
3. Cada possível sequência de $L(N) - 1$ dígitos deve ser usada ou como código ou ser prefixo de outros códigos.

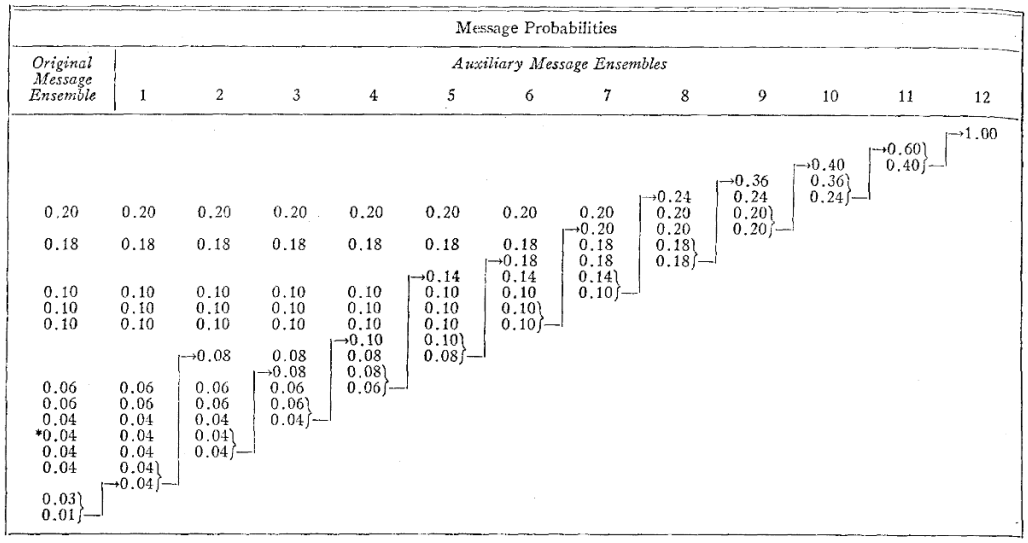
Em seguida, serão explicados os passos do algoritmo apresentado em Huffman (1952). Por fim, será mostrado um exemplo. Para facilitar, vamos utilizar o problema de codificação binária, $D = 2$, com valores possíveis do dígito 0 e 1.

A explanação partirá de que a tabela das probabilidades de cada símbolo já está pronta. Para gerar essa tabela, podem ser simplesmente contadas quantas vezes cada símbolo aparece em relação ao total dos símbolos da mensagem.

Tomando a tabela de probabilidades ordenada, os dois símbolos de menor probabilidade têm seus sufixos definidos por 0 e 1. Em seguida, esses símbolos podem ser considerados como apenas um, com probabilidade igual a soma das duas probabilidades. A tabela então é atualizada removendo-se os dois símbolos e adicionando o símbolo unido. O processo volta a ser executado, iterativamente, até que haja somente um símbolo que seja a união de todos, com probabilidade 1. Cada vez que é definido o sufixo, considere-se que é inserido a frente do sufixo anterior de cada símbolo o valor 0 ou 1.

O exemplo a seguir mostra o funcionamento do algoritmo de forma mais clara. Na figura abaixo, são mostrados os passos da codificação exemplificados por Huffman (1952).

Figura 6.1: Execução dos passos de codificação de Huffman.



Fonte: Huffman (1952)

Os códigos binários para cada símbolo gerados foram os seguintes:

Figura 6.2: Códigos binários gerados pela codificação de Huffman.

i	$P(i)$	$L(i)$	$P(i)L(i)$	*Code*
1	0.20	2	0.40	10
2	0.18	3	0.54	000
3	0.10	3	0.30	011
4	0.10	3	0.30	110
5	0.10	3	0.30	111
6	0.06	4	0.24	0101
7	0.06	5	0.30	00100
8	0.04	5	0.20	00101
9	0.04	5	0.20	01000
10	0.04	5	0.20	01001
11	0.04	5	0.20	00110
12	0.03	6	0.18	001110
13	0.01	6	0.06	001111
			$L_{av} = 3.42$	

Fonte: Huffman (1952)

O método apresentado é muito utilizado e possui variadas implementações na literatura. Além disso, serve de base para diversos compressores de arquivos, também estando presente em padrões de codificação de imagens, tal qual o JPEG, ou de codificação de vídeo, por exemplo, o H.264.

APÊNDICE B
Codificação de Vídeo

A comunicação através de vídeo, nas últimas décadas, tem sido uma das principais formas de comunicação, sob as mais diversas formas. Sullivan and Wiegand (2005) apresentam alguns cenários onde a vídeos digitais são aplicados:

- Transmissão televisiva via satélite, ou cabo;
- Serviços de conversação via internet;
- Streaming de vídeos;
- Gravação e armazenamento de vídeos, por exemplo DVD.

Se imaginar-se armazenar, por exemplo, apenas uma imagem RGB, 1.280×720, sem nenhuma compressão, utilizando 1 byte por canal de cor, a quantidade de bytes necessários para tal tarefa seria $1.280 \cdot 720 \cdot 3 = 2.764.800$ bytes, ou, aproximadamente, $2{,}7MB$ por imagem. Expandindo a ideia, um vídeo de 10 minutos, ou 600 segundos, utilizando uma taxa de 30 quadros por segundo, precisaria de quase $50GB$ para ser armazenado!

Esse exemplo ilustra a necessidade de codificar os vídeos de forma a reduzir substancialmente a quantidade de informação necessária para armazená-los. A partir disso, serão descritas algumas características de *codecs* (***co**der* e ***dec**oder* de vídeo) precisam observar.

Um dos princípios básicos utilizados na codificação de vídeos é a de que o olho humano é mais sensível ao brilho do que a cor diretamente (SULLIVAN; WIEGAND, 2005). Ou seja, a informação de crominância pode ser representada com menor precisão que a informação de luminância. De fato é isso que acontece na maioria dos *codecs*. Em geral, é utilizada o espaço de cor Y, Cb, Cr, no qual Y, chamado componente *lumma*, ou o brilho, em geral é codificado com dimensão 4 vezes maior que Cb e Cr, componentes *chroma* do cinza para o azul (Cb) e do cinza para o vermelho (Cr).

Em alguns *codecs* de vídeo existe a possibilidade de utilizar os formatos *progressive* e *interlaced* (SULLIVAN; WIEGAND, 2005). No caso de *progressive*, todo quadro é salvo em sequência. Já no formato *interlaced*, podem ser utilizados campos entrelaçados. Por exemplo, um primeiro campo pode ser as linhas pares do quadro, enquanto o segundo as linhas ímpares (SULLIVAN; WIEGAND, 2005). Na sequência do texto será utilizado o termo imagem sem diferenciação entre os formatos apresentados.

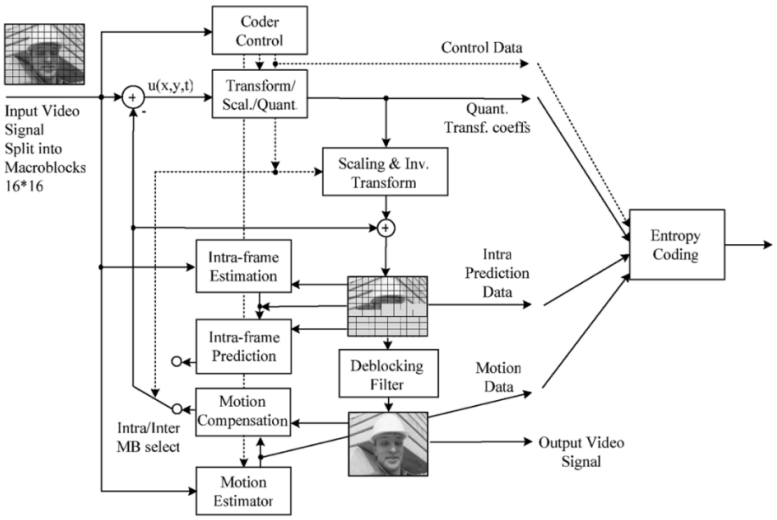
Técnicas para compressão digital, não somente para vídeos, mas para aplicações em geral, podem ser classificadas como segue (SULLIVAN; WIEGAND, 2005):

- **Predição:** Processo no qual são criadas predições sobre a informação que virá. Caso a predição seja de boa qualidade, a diferença entre a predição e a imagem em si, ou resíduo, será pequena e representável com menos bits.
- **Transformações:** Este processo é o de buscar outras formas de representar os dados, em geral mais esparsas (ou comprimidas). Por exemplo, para imagens a DCT fornece uma transformação mais compressível que a utilização da imagem original.
- **Quantização:** O processo de quantização se refere a representar os dados com menor quantidade de informação, ou seja, menos bits. Um exemplo é trocar uma representação em ponto flutuante por uma representação de inteiros com 8 bits.
- **Codificação de entropia:** Processo que utiliza a redundância de informação para criar uma tabela de representação na qual a informação mais redundante é codificada com menos bits e a mais, com mais bits. A codificação de Huffman é um exemplo desse processo.

Uma forma simples de fazer compressão de vídeos se dá por tomar cada imagem do vídeos separadamente e comprimi-la utilizando técnicas relativas a imagens, por exemplo JPEG (SULLIVAN; WIEGAND, 2005). Ainda podem ser utilizadas técnicas de predição intra-quadro, visando melhorar a codificação. Essas técnicas atingem certo grau de compressão, porém vídeos possuem grande correlação temporal entre seus quadros, ou seja, o quadro do instante t está fortemente correlacionado ao quadro $t - 1$ e ao quadro $t + 1$, em geral. Utilizar a forma anterior não aproveita esse fato, o que leva a se imaginar buscar metodologias que façam codificação inter-quadro.

A Figura 8 apresenta a ideia do funcionamento geral de um *encoder* de vídeo, mostrando, em alto nível, o funcionamento do mesmo.

Figura 6.3: *Encoder* de vídeo híbrido (em especial H.264).



Fonte: Sullivan and Wiegand (2005).

Um conceito que aproveita a dependência temporal é o MCP (*motion-compensated prediction*). O MCP considera que, muitas vezes, as mudanças durante os vídeos acontecem devido ao movimento. Dessa forma, se for possível estimar um vetor de movimento, seria possível fazer uma predição de compensação de movimento (SULLIVAN; WIEGAND, 2005).

Diversas formas de executar-se o MCP foram desenvolvidas pela literatura, unindo ganhos na complexidade a ganhos em compressibilidade e qualidade, utilizando uma ou mais imagens do passado como base para criar predições de movimento.

A seguir será explicado, de forma superficial, o funcionamento do codec H.264, com o qual serão feitas comparações nesse trabalho.

Codec H.264

O codec H.264 foi lançado como padrão em 2003. Ele incorporou funcionalidades de codecs anteriores, como H.262 e H.263, mas também trouxe novas funcionalidades.

Antes de entrar mais especificamente nas funcionalidades, é necessário entender-se o funcionamento geral do codec. Em primeiro lugar, podemos dividir o codec em *Network Abstraction Layer* (NAL), a parte relativa à configuração da rede pela qual o video será transmitido, e a *Video Coding Layer* (VCL), que diz respeito a forma como será representado o vídeo e seus quadros. Neste resumo o foco será nas funcionalidades relativas a codificação de vídeo, por tratarem de um assunto mais próximo a proposta do

trabalho.

Sendo assim, a seguir serão listadas funcionalidades presentes no codec. Basicamente, a VCL separa cada imagem em porções com uma quantidade variável de macroblocos de dimensão 16×16. Estes, por sua vez, podem ser divididos em sub-blocos de 16×8, 8×16, 8×8, 8×4, 4×8 e 4×4. A partir dos sub-blocos, dos macroblocos e das porções é que a codificação de vídeo é executada, levando em consideração o MCP.

Cada um dos quadros pode somente fazer predição intra-quadro (I-quadros), utilizar predição inter-quadros (P-quadros) com uma imagem de referência, ou utilizar duas imagens de referência, com pesos diferenciados (B-quadros). Segundo Wiegand et al. (2003), as principais funcionalidades relativas a eficiência da codificação são as seguintes:

- **Compensação de movimento (CM) com blocos de tamanho variável:** Dentro do macrobloco, cada um dos sub-blocos pode fazer CM.
- **CM com acurácia de um quarto de amostra:** A maioria dos codecs anteriores provia CM com acurácia de um meio de amostra. O codec H.264 aumenta o poder de acurácia para um quarto.
- **Vetores de movimento nas bordas da imagem:** Já presente como opcional no padrão H.263, permite que bordas também tenham vetores de movimento, utilizando técnicas de extrapolação das bordas.
- **Imagens de referência para CM múltiplas:** Permite que o *encoder* escolha entre múltiplas imagens as referências para fazer CM.
- **Possibilidade de utilizar imagens preditas como referência:** Em alguns codecs anteriores não era permitido utilizar imagens preditas como referência para o processo de CM. O H.264 remove essa restrição.
- **Predição com pesos:** Permite que o MCP atribua pesos e offsets aos sinais de referência utilizados.
- **Melhorias na predição intra-imagem:** Foram aprimoradas técnicas de extrapolação de arestas para a predição intra-imagem.
- **DCT aplicada a blocos menores:** No H.264 a DCT é aplicada a blocos de dimensão 4×4, utilizando uma transformada inteira. Porém é possível aplicar a DCT de mesma dimensão novamente a blocos maiores, de forma hierárquica.
- **Melhorias na codificação de entropia:** Foi melhorada a codificação de entropia utilizando um método chamado *arithmetic coding*.

Esse apêndice buscou apresentar os conceitos básicos utilizados na codificação de vídeos, dando enfoque principal ao padrão H.264, fortemente utilizado no meio.

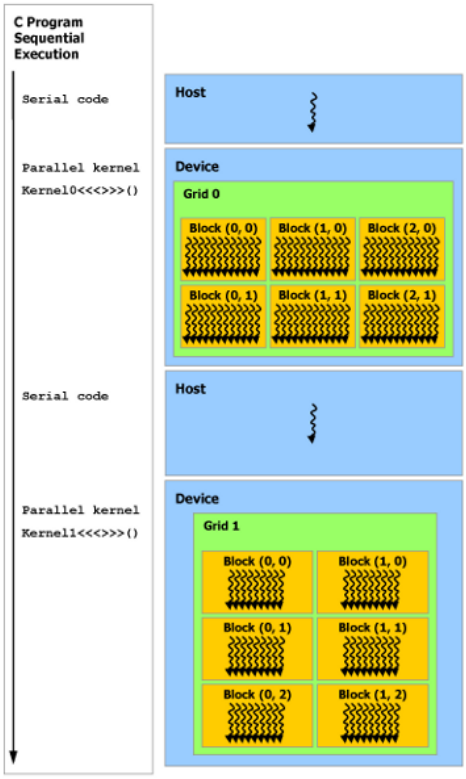
APÊNDICE C
Introdução a CUDA®

CUDA® é uma plataforma de programação paralela, bem como um modelo de programação, desenvolvido pela Nvidia para aplicações de propósito geral poderem ser executadas em GPU's. Sua primeira versão foi lançada em 2006. Desde então, diversas empresas e trabalhos acadêmicos utilizaram a plataforma. Além da forte utilização, também bibliotecas baseadas na plataforma surgiram, com o intuito de prover funções para aplicações já conhecidas como paralelizáveis. Exemplos de tais plataformas são cuFFT, para aplicações com transformada de Fourier, cuBLAS, para álgebra linear básica, NPP, para processamento de vídeo, imagens e sinais em geral.

A plataforma tem ótimos resultados para algoritmos, procedimentos, que possam utilizar fortemente o paralelismo (NVIDIA, 2018b). Atualmente, GPU's podem possuir mais de 4000 cores, por exemplo, a Nvidia Geforce RTX 2080 Ti, além de serem capazes de rodarem 32 threads simultaneamente. Todos os cores, ou pelo menos aqueles que forem designados para determinada aplicação, devem estar rodando o mesmo código, porém com valores de variáveis que podem ser diferentes.

A arquitetura de CUDA® se dá como na figura abaixo. Ao ser executado um determinado *kernel*, este é distribuído em um *grid* com B blocos, cada bloco com T threads. Dentro de um bloco pode ser compartilhada memória de rápido acesso, enquanto para acesso fora do bloco precisa-se de memória global.

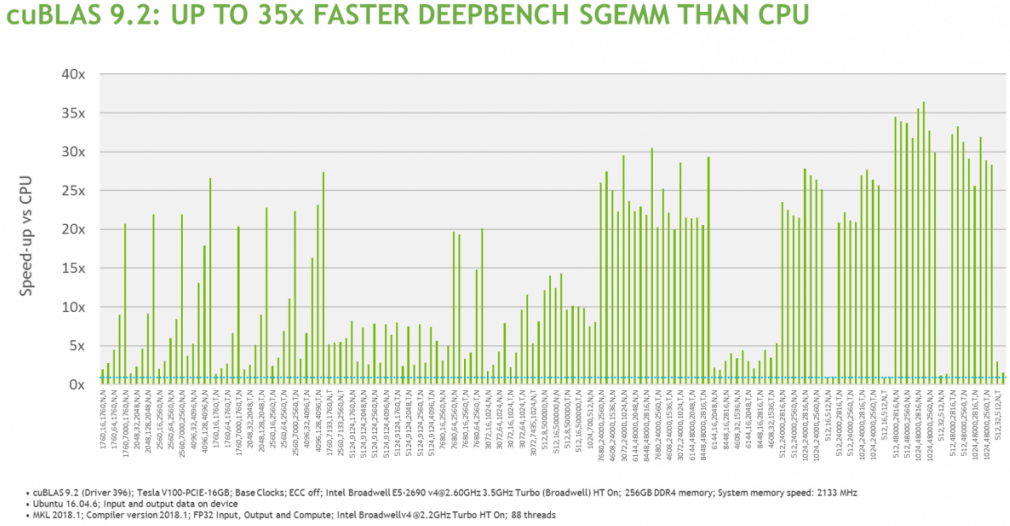
Figura 6.4: Arquitetura de CUDA®.



Fonte: (NVIDIA, 2018a)

Os resultados que CUDA® apresenta são muito consistentes para algoritmos claramente paralelizáveis, como é possível verificar no gráfico abaixo, que apresenta o *speedup* alcançado por funções da cuBLAS em comparação com as mesmas funções executadas na CPU.

Figura 6.5: Speedup de cuBLAS (GPU) em relação a MKL BLAS (CPU).



Fonte: (NVIDIA, 2018c)

Em geral o uso da interface com CUDA, através de alguma das linguagens de programação que permitem o uso, tais quais Python, C++, Fortran, se dá nas seguintes etapas:

1. Alocação inicial de memória da GPU;
2. Envio dos dados para a GPU;
3. Processamento das informações;
4. Aquisição dos dados;
5. Liberação de memória da GPU.

Os passos de alocação e liberação de memória demandam bastante tempo de CPU. Em geral, para ganhos de performance, tenta-se alocar memória somente no início da execução da aplicação, enviando, processando e recuperando as computações executadas sem gerar novas alocações.

Para exemplificar o quão simples é executar programas claramente paralelos em CUDA®, a seguir está descrito um *kernel* que soma duas matrizes, $\boldsymbol{A}$ e $\boldsymbol{B}$, e põe o resultado em $\boldsymbol{C}$.

```
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}
```

É possível perceber que, além da simplicidade do código, a sintaxe é muito próxima a da linguagem C. Esse fato pode ser considerado uma ajuda para a popularização da plataforma. Também é possível verificar como se dá o paralelismo. Através dos parâmetros x e y de $threadIdx$, seleciona-se qual parte da matriz que será somada. Nesse caso, por simplicidade, é considerado que o *kernel* será criado com apenas 1 bloco, de forma que somente é necessário utilizar o número da thread para definir os índices operados. Caso fossem utilizados mais blocos, deveriam ser considerados os valores de $blockIdx$, que corresponde ao identificador de que bloco está sendo processado, e $blockDim$, que representa o tamanho do bloco.