Universidade Federal de Pernambuco

Centro de Informática

Mestrado em Ciência da Computação

Codificação de Vizinhança para Compressão de Imagens e Reconhecimento de Forma

Tiago Buarque Assunção de Carvalho

Dissertação de Mestrado

Recife - PE

29 de março de 2010

Universidade Federal de Pernambuco

Centro de Informática

Tiago Buarque Assunção de Carvalho

Codificação de Vizinhança para Compressão de Imagens e Reconhecimento de Forma

Trabalho apresentado ao Programa de Mestrado em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Tsang Ing Ren*

Co-orientador: *George Darmiton da Cunha Cavalcanti*

Recife - PE

29 de março de 2010

Dedico este trabalho ao meu orientador Tsang Ing Ren

e ao meu co-orientador George Darmiton.

Agradecimentos

Agradeço a Tsang Ing Ren a Denise Jaeger Tenório pela prontidão na ajuda para a elaboração desse texto, aos meus familiares e amigos pela compreensão, aos meus professores e colegas pelo incentivo e força, e a Maria pela árdua função de me acalmar.

Tudo o que o tempo tem dado
De tempo em tempo se soma
E o tempo com o tempo toma
Tudo o que deu no passado

—SIBA E A FULORESTA (Toda Vez que eu Dou um Passo o Mundo Sai do Lugar, 2007)

Resumo

A codificação de vizinhança foi inicialmente proposta em [TTD99]. Essa codificação gera um conjunto de códigos de vizinhança para representar uma imagem binária. Uma das principais limitações da proposta inicial é que a codificação não era capaz de reconstruir a imagem representada por ela. Esta dissertação oferece uma nova abordagem para a codificação de vizinhança, que permite representar imagens binárias sem perdas, possibilitando a reconstrução da imagem codificada. Outro problema da codificação de vizinhança é o excessivo e redundante número de elementos no conjunto de códigos gerados para cada imagem. Para resolver este problema, são propostos, no presente trabalho, três algoritmos para a redução do conjunto desses códigos. Um destes algoritmos de redução do conjunto de códigos segue uma abordagem evolucionária. Outra contribuição realizada aqui diz respeito à representação de vizinhanças em um código de vizinhança. Esta representação é construída através de funções braços. Novas funções braços podem ser definidas facilmente para representar diferentes vizinhanças. Esta maneira de abordar vizinhanças possibilita o reuso dos algoritmos propostos sem necessidade de adaptação. Ainda são propostas duas aplicações utilizando codificação de vizinhança: compressão de imagens binárias e reconhecimento de forma. O método de compressão proposto baseia-se na redução do número de códigos necessários para reconstruir a imagem sem perdas. O reconhecimento de forma é avaliado para o problema de recuperação de imagens semelhantes.

Palavras-chave: Codificação de imagens, representação, compressão, imagem binária, algoritmos genéticos, análise de forma, classificação.

Abstract

Neighborhood coding was first proposed at [TTD99]. That encoding creates a set of neighborhood codes which represents a binary image. The main lack of the initial proposed coding it that is impossible to reconstruct the represented image. This dissertation offers a new approach for the neighborhood coding that makes lossless representation of binary images providing a way of fully reconstructing the encoded image. Another lack of the neighborhood coding is the huge and redundant number of elements in the set of codes generated for each image. In order to solve that problem three algorithms for reducing the number of element in set of codes are proposed on this work. One of those three proposed algorithm employs a genetic algorithm. Another contribution of this dissertation is related to the neighborhood representation in a neighborhood code. This representation is built through segment functions. New different segment functions can be defined for representing a variety of neighborhoods. This approach for the neighborhood representation allows reuse the proposed algorithms without any adaptation. The neighborhood coding is tested in two applications: image compression and shape recognition. The compression method is based on the reduction of the set of neighborhood codes needed to reconstruct the image without loss. The shape recognition is evaluated for the task of recover similar images.

Keywords: Image coding, representation, compression, binary (bi-level) image, genetic algorithm, shape analysis, classification.

Sumário

1 Introdução **1**

1.1 Objetivos e Contribuições 4

1.2 Organização do Documento 5

2 Representação, Compressão e Reconhecimento de Imagens Binárias **7**

2.1 Representação de Imagens 7

2.1.1 Representação por Bitmap 8

2.1.2 Representação Interna 8

2.1.3 Representação de Contorno 10

2.2 Métodos de Compressão de Imagens Binárias 11

2.2.1 CCITT Group3 e Group4 11

2.2.2 GIF e PNG 14

2.2.3 JBIG e JBIG2 16

2.3 Reconhecimento de Forma 19

3 Código de Vizinhança **24**

3.1 Imagem Binária como Matriz de Bits 25

3.2 Codificação de Vizinhança 27

3.3 Funções Braços – Vizinhanças 28

3.3.1 Funções Braços Horizontais e Verticais 31

3.3.2 Funções Braços Diagonais 32

3.4 Código de Vizinhança (XC) 34

3.5 Configurações de XC 36

3.5.1 XC Tipo Torre 38

3.5.2 XC Tipo Bispo 39
3.5.3 XC Tipo Rainha 39
3.6 Reconstrução da Imagem a partir da Codificação de Vizinhança 40
3.7 Geração da Codificação de Vizinhança a partir da Imagem Binária 41
3.8 Algoritmos para Redução do Conjunto de Códigos 44
3.8.1 Redução do Conjunto de Códigos Via Algoritmo Genético (REDAG) 44
3.8.2 Primeiro Algoritmo Determinístico para Redução do Conjunto de Códigos (RED1) 52
3.8.3 Segundo Algoritmo Determinístico para Redução do Conjunto de Códigos (RED2) 53
3.9 Experimentos 57
3.9.1 Base de Imagens para Teste 57
3.9.2 Testes com o REDAG 59
3.9.3 Testes com o RED1 60
3.9.4 Testes com o RED2 61
3.9.5 Considerações Finais 62

4 Compressão de Imagens com Código de Vizinhança 67
4.1 Método de Compressão 68
4.2 Codificação Huffman 72
4.3 Formato do Arquivo 76
4.4 Experimentos 78

5 Reconhecimento de Forma com Codificação de Vizinhança 82
5.1 Padronização de Rotação 83
5.2 Distância entre Conjuntos de Códigos 87
5.3 Distância entre Duas Formas 90
5.4 Experimentos 92

6 Conclusão e Trabalhos Futuros 93

Lista de Figuras

1.1 (a) Exemplo de imagem em tons de cinza. (b) Exemplo de imagem binária halfone. Esta imagem é uma versão *halftone* da imagem (a) gerada a partir do processo de Floyd-Steinberg. 2

1.2 (a) Exemplo de imagem binária de texto, esta figura possui tanto texto gerado por computador com texto manuscrito digitalizado. (b) Exemplo de imagem binária de desenho. 3

2.1 Exemplos de esqueletizações [Dou94]. O círculo é representado apenas por um ponto. O quadrado é representado por eixos que cruzam suas diagonais. As linhas pretas marcam os esqueletos do retângulo e da coroa. 9

2.2 Exemplo de decomposição por quadtree. Cada região é dividida em quatro novas regiões até que todas as regiões se tornem uniformes. 9

2.3 Exemplos de códigos conectividades usados em [SCBRD07]: (a) 3OT, direções ortogonais de cadeias de três símbolos (*orthogo- nal chain directions of three symbols*); (b) F8, cadeia de Freeman de oito direções; (c) F4, cadeia de Freeman de quatro direções. 10

2.4 Exemplo de codificação de cadeia [GW10] de um contorno usando a conectividade F8 (cadeia de Freeman de oito direções) da Figura 2.3 (b). 10

2.5 Exemplo da codificação MH (*Modified Huffman*) empregada pelo padrão Group 3. A Figura (a) é uma linha de uma imagem binária codificada pelo algoritmo MH. Cada seqüência de pixels da mesma cor é representada por um código que indica seu tamanho. Cada código está definido na tabela do MH, as doze primeiras linhas dessa tabela estão exibidas na Figura (b). A Figura (a) inicia com um código indicando que o número de pixels brancos no começo da linha é zero pois o MH assume que o primeiro pixel de cada linha é branco. 13

2.6 Exemplo da execução do algoritmo LZW. A cadeia de entrada (representada por letras) é convertida numa cadeia de índices para instâncias do dicionário. A cada índice inserido na cadeia de saída, uma nova instância é adicionada ao dicionário. 15

2.7 Exemplo do funcionamento do algoritmo gZip, uma série de caracteres é substituída por um ponteiro para a anterior, sob a forma de uma tupla (distância, comprimento). 16

2.8 Exemplo de várias resoluções de uma mesma imagem. As duas imagens da esquerda apresentam maior resolução e formam as camadas diferenciais do JBIG. A imagem da direita apresenta menor resolução e forma a camada base do JBIG. 17

2.9 *Templates* da camada base. As bolas cinzas são os pixels a serem codificados, as pretas e brancas são pixels já codificados e o *alvo* é o pixel atual. As bolas pretas representam os pixels que fazem parte do *template*. 18

2.10 Exemplos da base de imagens binárias de silhuetas de objetos *Core Experiment CE-Shape-1* parte A. A linha de cima contém imagens da base A1, que contém em cada classe uma mesma imagem em escalas diferentes. A linha de baixo contém imagens da base A2, que contém, por classe, uma mesma imagem em diferentes rotações. 21

2.11 Exemplos da base de imagens binárias silhuetas de objetos *Core Experiment CE-Shape-1* parte B. A linha de cima contém 3 das 20 imagens da classe DEER. E a linha de baixo contém 3 das 20 imagens da classe DEVICE. Percebe-se que os elementos de cada classe apesar de semelhantes representam formas de objetos distintos. 21

2.12 Exemplos da base de imagens binárias silhuetas de objetos *Core Experiment CE-Shape-1* parte C. A linha de cima da figura contém quadros do vídeo de um peixe nadando, a imagem mais à esquerda da primeira linha primeira dos 200 quadros do vídeo. A linha de baixo contém exemplos das 1100 imagens de outros peixes e animais marinhos. 22

3.1 Vetor de vizinhança proposto por [TTD99]. A grade é conjunto de pixels pretos da imagem. Os quadrados cinza são os pixels representados pelo vetor de vizinhança V = (5,2,9,4) – 5 pixels na direção direção Norte, 2 na direção Leste, 9 na direção Sul e 4 na direção Oeste, $V = (n,l,s,o)$. 25

3.2 Exemplo de um código de vizinhança numa imagem binária. (a) Exemplo de uma imagem binária de dimensões 12×7, as células cinza da grade representam os pixels pretos e as demais células representam os pixels brancos; (b) É destacado um código de vizinhanças da imagem (a), este XC é descrito por $(x = 8, y = 3, n = 2, l = 2, s = 2, o = 7)$. 28

3.3 Codificação de Vizinhança. As Figuras de (a) a (i) destacam os 9 códigos de vizinhança da imagem (j), cada código é representado por um vetor da forma (x,y,n,l,s,o). O centro de cada código está destacado na imagem em preto e as vizinhanças em cinza escuro. Um XC é gerado para cada pixel preto da imagem. Códigos das imagens: a = (2,1,0,0,3,0, b = (1,2,0,2,0,0), c = (2,2,1,1,2,1), d = (3,2,0,0,1,2), e = (2,3,2,1,1,0), f = (3,3,1,0,0,1), g = (1,4,0,1,0,0), h = (2,4,3,0,0,1), i = (3,5,0,0,0,0). 29

3.4 Exemplo de vetores XC torre (a), bispo (b) e rainha (c). A célula preta da grade é o centro do XC, as células cinza escuro são os pontos representados por cada XC e as cinza claro são os demais pontos de pixels da imagem, as células brancas são os pixels do fundo. O vetor XC $\varphi^{torre} = (8,4,1,3,5,6)$ de (a) é do tipo torre. O vetor XC $\varphi^{bispo} = (8,4,2,3,5,2)$ de (b) é do tipo bispo. O vetor XC $\varphi^{rainha} = (8,4,1,2,3,3,5,5,6,2)$ de (c) é do tipo rainha. 38

3.5 *Crossover* de 1 ponto. Um ponto j do cromosso é escolhido randomicamente, o vetor do cromossomo é dividido em dois segmentos: antes e depois do ponto j. Dois cromossomos pais que tiveram seus vetores divididos no ponto j trocam o segundo de seus segmentos dando origem a dois novos indivíduos chamados filhos. 49

3.6 Exemplo de mutação. Cada gene do vetor cromossomo de um filho tem seu mudado com uma probabilidade de mutação p_m gerando um filho' diferente do inicial. 50

3.7 O nome de cada imagem do conjunto de testes e a escala que cada uma aparece nesta imgem. Quando a escala é 0,5× indica que as dimensões da imagem foram reduzidas para a metade. (a) *0*, 0,5×; (b) *1*, 0,5×; (c) *2*, 0,5×; (d) *3*, 0,5×; (e) *4*, 0,5×; (f) *5*, 0,5×; (g) *6*, 0,5×; (h) *7*, 0,5×; (i) *8*, 0,5×; (j) *9*, 0,5×; (k) *a*, 2×; (l) *bell-2*, 0,5×; (m) *bat-12*, 0,25×; (n) *grafico*, 0,25×; (o) *cat*, 0,175×; (p) *ouster*, 0,5×; (q) *courier12*, 0,5×; (r) *times12i*, 0,5×; (s) *oldeng16*, 0,5×; (t) *wingdings72-1*, 0,5×; (u) *wingdings72-2*, 0,5×; (v) *wingdings72-3*, 0,5×; (w) *retangulos-size1*, 0,5×; (x) *retangulos-size2*, 0,5×. 58

4.1 Diagrama de fluxo do método de compressão propostos. O processo é dividido em seis etapas: reduzir o conjunto de códigos, agrupar os códigos por vizinhança, separar a vizinhança dos centros, aplicar RLE às vizinhanças, aplicar RLE aos contadores do outro RLE e, por último, aplicar a codificação Huffman. 68

4.2 Uma imagem binária e os códigos XC tipo Torre que a representam. Os códigos riscados foram eliminados por algum algoritmo de redução de códigos. Com os códigos restantes ainda é possível reconstruir a imagem sem perdas 69

4.3 Exemplo da construção da árvore da codificação Huffman. Em (a) cada um dos símbolos está associado a sua freqüência. Em (b) os símbolos de menor freqüência são agrupados em um nó, a freqüência do nó e a soma das freqüências dos objetos agrupados. Em (c) o novo nó agrupa os objetos de menor freqüência que ainda não estão agrupados. O processo continua em (d) e a árvore está completamente construída em (e). 75

4.4 Ordem de leitura da árvore Huffman gerada pelo Algoritmo 4.1. A árvore é percorrida em largura, isto é, os nós do próximo nível só são visitados depois que todos os nós do nível anterior a ele já foram visitados. Esta ordem de leitura define como a árvore é escrita no arquivo. 77

4.5 Representação binária da árvore descrita na Figura 4.4. O primeiro trecho é uma representação binária da árvore que usa bits 1 para representar um nó e bits 0 para representar uma folha. *s* tem tamanho fixo de 4 bits e representa o número de bits usado para representar o símbolo de cada folha da árvore. Cada símbolo d, e, f, h, i, é presentado por *s* bits. 77

5.1 Diagrama de fluxo do método de reconhecimento forma usando codificação de vizinhança. A padronização de rotação rotaciona uma imagem para uma forma canônica. A extração de códigos é realizada gerando um código para cada pixel preto da imagem e o conjunto de códigos é reduzido pelo algoritmo RED2. A distância entre duas imagens é calculada comparando cadeias de códigos XC através de *template matching*. 83

5.2 Exemplos da padronização de rotação. Em cada linha as imagens estavam originalmente rotacionadas em 0, 9^o, 36^o, 45^o, 90^o e 150^o. Na primeira linha ocorreu um problema de espelhamento da imagem canônica em relação ao eixo horizontal. E na segunda linha metade das imagens canônicas apareceram rotacionadas em 180 graus. 88

5.3 Cálculo da distância de edição para cadeias de caracteres. O custo de inserção, remoção e substituição é igual a 1 para qualquer caractere: (a) uma inserção; (b) uma substituição; (c) uma remoção; (d) duas cadeias idênticas não precisam de inserção, remoção ou substituição para se tornarem iguais, portanto a distância entre elas é zero. Figura retirada de [TK06]. 89

Lista de Tabelas

2.1 Taxas de acerto percentual dos descritores P298, P320, P517, P567 e P687sobre as bases *Core Experiment CE-Shape-1* A1, A2, B e C. 23

3.1 Dimensões das imagens do conjunto de teste. *Largura* é o número de pixel na horizontal ou o número de colunas da imagem. *Altura* é o número de pixels na vertical ou o número de linhas da imagem. 63

3.2 Redução do conjunto de códigos XC através do algoritmo REDAG. *Total* é o número de código antes da execução do REDAG. *Mínimo, Máximo, Mediana, Média* e *Desvio Padrão* são referentes ao tamanho do conjunto de códigos reduzido gerado pelo REDAG após 100 repetições para cada imagem. 64

3.3 *Mediana* e *Média* do número de gerações necessárias para encontrar o melhor resultado em cada repetição do REDAG. Estes resultados são referentes às mesmas execuções da Tabela 3.2. 65

3.4 Resultado da redução do conjunto de códigos XC. *Total* é o número de códigos antes da redução. O resultado do *REDAG* é o melhor entre 100 repetições, ver Tabela 3.2. *RED1T, RED1B, RED1R, RED2T, RED2B* e *RED2R* são os resultados da redução usando RED1 e RED2 com os códigos XC Torre (T), Bispo (B) e Rainha (R), respectivamente. 66

4.1 Tabela de símbolos numéricos e suas respectivas freqüências em uma determinada cadeia. A coluna da direita mostra o código gerado para cada. 73

4.2 Resultados do método de compressão proposto. Os resultados estão expressos em bpp (bits por pixel). Os métodos de redução testados foram o REDAG com a configuração Torre, o RED1 com a configuração Torre (RED1T), o RED1 com a configuração Bispo (RED1B), o RED1 com a configuração Rainha (RED1R), o RED2 com a configuração Torre (RED2T), o RED2 com a configuração Bispo (RED2B) e o RED2 com a coniguração Rainha (RED2R). Os resultados dessa tabela foram medidos no mesmo experimento que gerou os dados da Tabela 3.4. Os resultados para o REDAG são os melhores entre 100 execuções do algoritmo. O símbolo * indica o melhor resultado da compressão. O símbolo ** indica o segundo melhor resultado da compressão. 80

4.3 Comparação entre os métodos de compressão propostos usando a redução de códigos RED2 com as conigurações Torre (RED2T) e Rainha (RED2R) e os formatos CCITT Group 3 e Group 4, PNG, GIF e JBIG. Os resultados estão expressos em bpp (bits por pixel). 81

5.1 Taxas de acerto no reconhecimento de forma utilizando codificação de vizinhança (método proposto), sob a base *Core Experiment CE-Shape-1* A2, comparados com os algoritmos P298, P320, P517, P567 e P687, descritos na Seção 2.3. 92

Lista de Algoritmos

3.1 Reconstruindo a Imagem a partir da Codificação de Vizinhança 42

3.2 Gerando a Codificação de Vizinhança a partir de uma Imagem Binária 45

3.3 REDAG: Redução do Conjunto de Códigos XC via Algoritmo Genético. 51

3.4 RED1 54

3.5 RED2 55

4.1 Gerando a árvore da codificação Huffman. 73

5.1 Calculando a distância entre duas cadeias de códigos XC. 91

Lista de Símbolos

Símbolo	Descrição	Página
P	representação matricial de uma imagem binária	25
p_{xy}	valor de um pixel numa imagem binária na posição (x,y)	25
(x,y)	posição de um pixel p_{xy} numa imagem binária	25
h	altura ou número de linhas de uma imagem	26
w	lagura ou número de colunas de uma imagem	26
$b(.)$	função braço	29
Π	conjunto de todas as posições dos pixels pretos de uma imagem binária	30
B	conjunto de todas as funções braço $b(.)$	30
$b_N(.)$	função braço norte	31
$b_L(.)$	função braço leste	31
$b_S(.)$	função braço sul	32
$b_O(.)$	função braço oeste	32
$b_{NE}(.)$	função braço noroeste	33
$b_{SE}(.)$	função braço sudeste	33
$b_{SO}(.)$	função braço sudoeste	33
$b_{NO}(.)$	função braço nordeste	34
XC	código de vizinhança	34
φ	vetor XC (código de vizinhança)	35
β	vetor de configuração do XC	35
u	função vizinhança de um XC	35
Θ	conjunto das funções braço utilizadas	36

Símbolo	Descrição	Página
φ^{torre}	vetor vizinhança XC tipo torre	38
β^{torre}	configuração XC torre	38
u^{torre}	função vizinhança torre	39
φ^{bispo}	vetor vizinhança XC tipo bispo	39
β^{bispo}	configuração XC bispo	39
u^{bispo}	função vizinhança bispo	39
φ^{rainha}	vetor vizinhança XC tipo rainha	40
β^{rainha}	configuração XC rainha	40
u^{rainha}	função vizinhança rainha	40
v	função vizinhança para um conjunto de códigos de vizinhança XC	41
AG	Algoritmo Genético	44
REDAG	AG para redução do conjunto de XCs	44
α	vetor cromossomo do REDAG	46
g_i	um gene do vetor cromossomo α	46
δ	vetor de códigos XC associado a α	46
Φ_{δ}^{α}	o conjunto de códigos representado por α em relação a δ	47
d	distância de Hamming entre duas imagens binárias	47
f	função de aptidão do REDAG	47
p_m	probabilidade de mutação	48
RED1	Primeiro Algoritmo Determinístico para Redução do Conjunto de Códigos	54
RED2	Segundo Algoritmo Determinístico para Redução do Conjunto de Códigos	55
bpp	bits por pixel	67
RLE	*Run Length Encoding*	70

CAPÍTULO 1

Introdução

As primeiras aplicações usando imagens digitais surgiram na década de 1920. Imagens eram transmitidas através de um cabo submarino entre Londres e Nova Iorque. Desta forma, o tempo de transmissão de uma imagem diminuiu de algumas semanas para menos de três horas [GW10]. No final da década de 1970, foi criado o primeiro padrão para transmissão de imagens binárias digitais, o CCITT Group 3, que é até hoje empregado em aparelhos de fax [Bar06, WMB99]. Desde então, as imagens binárias digitais passaram a ter um papel muito importante na conservação e transmissão da informação, principalmente depois da popularização do computadores. Esta dissertação comtempla representação, compressão e reconhecimento de imagens binárias. Este capítulo situa as imagens binárias entre os outros tipos de imagens digitais e demonstra, através de exemplos, a importância das imagens binárias digitais na sociedade contemporânea.

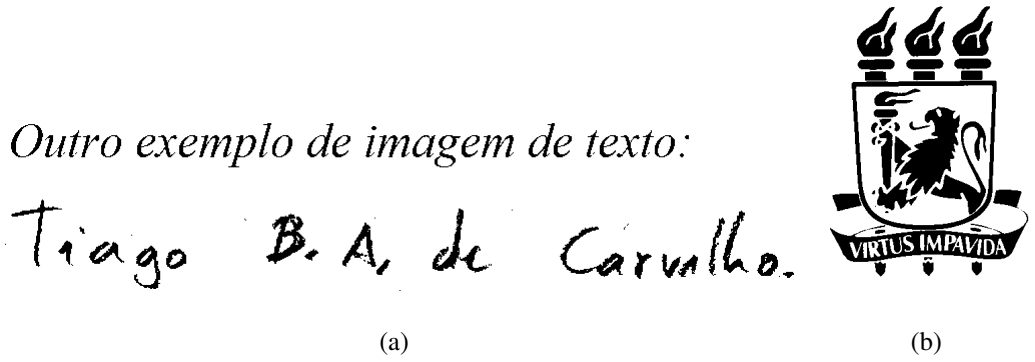
As imagens digitais são aquelas que podem ser representadas por elementos discretos. Essas imagens são manipuladas por computadores e aparelhos eletrônicos. As imagens digitais podem ser vetorizadas ou no formato *bitmap*. Imagens vetorizadas são descritas através de funções matemáticas empregadas para representar linhas, polígonos, objetos geométricos simples, além de texto. Imagens de *bitmap* são compostas por uma matriz de pixels, onde cada pixel representa uma cor. *Bitmaps* são usadas para representar vídeos ou imagens estáticas (fotos, desenhos, mapas etc.) [MR96]. Os *bitmaps* de imagens estáticas podem ser diferenciados pela forma como representam a cor de cada pixel, podendo ser classificados em imagens binárias, imagens em tons de cinza e imagens coloridas [GW10]. Se apenas um bit é usado para representar a cor de cada pixel, o *bitmap* representa uma imagem binária, onde apenas duas cores são possíveis. Os principais tipos de imagens binárias são: de documentos e de formas. Quando a cor de cada pixel é representada em uma escala de valores, o *bitmap* representa uma imagem em tons de cinza ou multitonal. Este tipo de imagem é suficiente para representar

(a) (b)

Figura 1.1 (a) Exemplo de imagem em tons de cinza. (b) Exemplo de imagem binária halfone. Esta imagem é uma versão *halftone* da imagem (a) gerada a partir do processo de Floyd-Steinberg.

fotografia em escalas de cinza. Um exemplo deste tipo de imagem pode ser visto na Figura 1.1(a). Imagens coloridas representam a cor de cada pixel através de um vetor. Exemplos desse tipo de imagens são fotografias e desenhos coloridos.

Imagens binárias, por sua vez, podem ser classificadas em vários tipos, por exemplo, imagens de desenho, imagens de texto e imagens *halftone*. Imagens binárias de texto podem conter tanto texto gerado por computador como texto manuscrito digitalizado, um exemplo de imagem binária de texto pode ser visto na Figura 1.2(a). Imagens de desenho são geralmente formadas por linhas e áreas contínuas de pixels pretos ou brancos, um exemplo de imagem binária de desenho pode ser vista na Figura 1.2(b). Diferentemente das imagens de desenho, que possuem os pixels de mesma cor concentrados em certas áreas da imagem, as imagens *halftone* possuem os pixels de mesma cor dispersos, simulando várias tons de cinza na imagem, porém é uma imagem binária. Um exemplo de imagem binária *halftone* é mostrado na Figura 1.1(b). Essa imagem foi gerada a partir da imagem em tons de cinza da Figura 1.1(a), o processo de Floyd-Steinberg [LA00] foi utilizado para a transformação da imagem em tons de cinza na imagem *halftone*. As imagens *halftone* são amplamente utilizadas na impressão de fotos e Figuras



(a) (b)

Figura 1.2 (a) Exemplo de imagem binária de texto, esta figura possui tanto texto gerado por computador com texto manuscrito digitalizado. (b) Exemplo de imagem binária de desenho.

[MV02].

Uma das áreas onde imagens binárias digitais são essenciais é na codificação de documentos, uma vez que a maior parte do conteúdo de um documento digital são imagens binárias – com exceção de documentos históricos que são representados com o máximo de informação possível [Pai10]. A importância desse tipo de imagens pode ser observada na digitalização de livros. Atualmente não só os livros começam a ser produzidos já em forma digital como existe um grande esforço em digitalizar a bibliografia já existente, isto pode ser visto através das chamadas bibliotecas digitais [Pai10, Les05]. Os livros no formato eletrônico são chamados *ebooks*, o quais têm a vantagem de poder ser distribuídos de forma rápida e barata, além de não precisarem ser impressos, reduzindo o custo também da produção. Os *ebooks* têm facilitado áreas como educação à distância [Gal04]. Através de novos dispositivos para a visualização de *ebooks*, como o Amazon Kindle, o Sony Reader e o iPad, os livros impressos começam a ser substituídos pelo *ebook* [JK10]. Os *ebooks* podem apresentar funcionalidades, como facilidade na busca, que vêm transformado o modo como os leitores usam os livros [NG08, PDF08].

Um desafio, que surgiu com o padrão Group 3 nos anos 70, é a compressão de imagens binárias. A função da compressão é representar a mesma imagem (compressão sem perdas) ou uma aproximação dela (compressão com perdas) usando menos bytes. A compressão reduz o custo de armazenamento e transmissão de imagens digitais. O primeiro método de compressão para imagens binárias digitais foi o MH (*Modified Huffman*) empregado no padrão Group 3.

Mais tarde, o mesmo padrão, passou a utilizar o método MR (*Modified READ*) para alcançar maior compressão, então foi criado o método MMR (*Modified MR*) em 1984 com a criação do padrão Group 4. Taxas maiores de compressão vem sendo alcançadas com novos métodos de compressão. Atualmente os formatos de imagem JBIG e DjVu detêm as maiores taxas de compressão para imagens binárias digitais. Informações sobre todos esses métodos podem ser encontradas em [WMB99, Les05, Bar06, Gla07].

No livro Processamento de Imagens Binárias Digitais (*Binary Digital Image Processing*) [MMS00], além de técnicas para processamento de imagens, estão descritas algumas aplicações que utilizam imagens binária digitais: (1) converter um desenho feito à mão de um circuito elétrico em uma representação vetorizada; (2) converter uma versão impressa de um mapa rodoviário em uma modelagem computacional que permite calcular distâncias e rotas; (3) reconhecimento de impressões digitais para identificação biométrica; (4) reconhecimento automático de texto manuscrito; (5) compressão de imagens.

Reconhecimento de forma é um problema que ganhou grande importância com a definição dos padrões MPEG-4 e MPEG-7. Estes padrões foram criados para, entre outras finalidades, a representação e a busca de imagens em conteúdo multimídia. Imagens binárias são usadas para representar formas nestes dois padrões [LLE00, JBB$^+$98, Mar04, Par02].

1.1 Objetivos e Contribuições

Os objetivos desse trabalho são: estender e formalizar o conceito de codificação de vizinhança para representação de imagens binárias, propor aplicações de compressão de imagens e reconhecimento de forma usando a codificação proposta, além de propor novas pesquisas utilizando tal codificação.

A codificação de vizinhança foi inicialmente proposta em [TTD99]. Essa codificação gera um conjunto de códigos de vizinhança para representar uma imagem binária. Uma das principais limitações da proposta inicial é que a codificação não era capaz de reconstruir a imagem representada por ela. Esta dissertação oferece uma nova abordagem para a codificação de vizi-

nhança, que permite representar imagens binárias sem perdas, possibilitando a reconstrução da imagem codificada.

Outro problema da codificação de vizinhança é o excessivo e redundante número de elementos no conjunto de códigos gerados para cada imagem. Para resolver este problema, são propostos, no presente trabalho, três algoritmos para a redução do conjunto desses códigos. Um destes algoritmos de redução do conjunto de códigos segue uma abordagem evolucionária.

Outra contribuição realizada aqui diz respeito à representação de vizinhanças em um código de vizinhança. Esta representação é construída através de funções braços. Novas funções braços podem ser definidas facilmente para representar diferentes vizinhanças. Esta maneira de abordar vizinhanças possibilita o reuso dos algoritmos propostos sem necessidade de adaptação.

Ainda são propostas duas aplicações utilizando codificação de vizinhança: compressão de imagens binárias e reconhecimento de forma. O método de compressão proposto baseia-se na redução do número de códigos necessários para reconstruir a imagem sem perdas. O reconhecimento de forma é avaliado para o problema de recuperação de imagens semelhantes. Algumas das contribuições deste trabalho já encontram-se divulgadas na comunidade acadêmica [dCTT+10] através da conferência *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

1.2 Organização do Documento

O restante do documento é organizado como segue: o Capítulo 2 é uma revisão sobre métodos de representação e compressão de imagens binárias, e reconhecimento de forma. O Capítulo 3 trata sobre a codificação de vizinhança. Neste capítulo, a codificação de vizinhança é formalizada e são propostos algoritmos para a extração de códigos de vizinhança e reconstrução de uma imagem binária a partir da codificação de vizinhança. No mesmo capítulo, são propostos e avaliados três algoritmos para a redução do conjunto de códigos gerados pela codificação de vizinhança. O conjunto de códigos reduzidos, gerado por cada algoritmo, é capaz de representar

a mesma imagem sem perdas. No Capítulo 4, é proposto e avaliado um método de compressão de imagens binárias usando codificação de vizinhança. No Capítulo 5, a codificação de vizinhança é proposta e avaliada como descritor de forma para o problema de reconhecimento de forma. O Capítulo 6 apresenta as conclusões desta dissertação e propostas de trabalhos futuros.

CAPÍTULO 2

Representação, Compressão e Reconhecimento de Imagens Binárias

O presente trabalho propõe a codificação de vizinhança para compressão de imagens binárias e reconhecimento de formas. Este capítulo é dividido em três seções. A codificação de vizinhança é um método de representação de imagens, portanto a primeira seção revisa alguns outros métodos de representação de imagens. A segunda seção revisa métodos de compressão de imagens. Enquanto a terceira seção faz uma explanação sobre o problema de reconhecimento de formas e algumas das técnicas utilizadas nessa área.

2.1 Representação de Imagens

Este trabalho utiliza as expressões representação de imagens e codificação de imagens de forma alternada, entendendo que elas têm o mesmo significado. Uma codificação de imagens é uma técnica usada para descrever uma imagem digital que tem duas abordagens principais: compressão e descrição de conteúdo [Par02]. O objetivo da compressão é economizar recursos de armazenamento e transmissão de dados. Enquanto a descrição de conteúdo visa indexar e recuperar imagens, um exemplo disso é o reconhecimento de formas.

A codificação de uma imagem pode ser realizada de duas maneiras distintas: com perdas, onde é representada uma aproximação da imagem original, e sem perdas, onde a imagem representada é exatamente igual à imagem original. Os métodos de segunda geração para codificação de imagens [RMB97] baseiam-se em características do sistema visual humano. Essa abordagem permite que a imagem seja representada com perdas em troca de uma maior taxa de compressão. A codificação de vizinhança é um método de representação de imagens sem

perdas, portanto nesta seção só serão revisados métodos de representação sem perdas.

A codificação de imagens binárias tem sido estudada nos últimos anos sob o problema de representação de formas. As três maiores classes de representação de formas são representação por *bitmap*, representação interna e representação de contorno [JBB+98]. Cada uma dessas classes é explanada a seguir.

2.1.1 Representação por Bitmap

As técnicas de representação por *bitmap* operam diretamente sobre os dados, isto é, operam diretamente na matriz de bits sem usar qualquer representação intermediária. Entre elas se destacam a MH (*Modified Huffman*), MR (*Modified READ*), MMR (*Modifiead MR*) e CAE (*Context-based Arithmetic Encoding*).

As técnicas MH, MR e MMR são usadas nos algoritmos de compressão CCITT Group 3 e Group 4. Ambos os algoritmos são explicados na Seção 2.2.1. A idéia de cada um desses métodos é representar uma seqüência de pixels de uma mesma cor, em cada linha da imagem, por uma cadeia de bits menor do que aquela que está sendo representada.

O CAE codifica cada pixel da imagem em função daqueles que já foram codificados anteriormente. Para tanto, faz uma predição do próximo valor do pixel a partir de um conjunto de *templates*. Este método é usado no algoritmo de compressão JBIG (seção 2.2.3).

2.1.2 Representação Interna

A representação interna visa representar regiões da imagem, opondo-se à representação do contorno. Duas técnicas principais podem ser destacadas nessa classe de representação: esqueletização [Dou94] e decomposição por *quadtree* [JBB+98]. A primeira extrai um esqueleto da imagem enquanto a segunda divide suas regiões de forma hierárquica.

A esqueletização representa uma imagem inteira através de um esqueleto. Esse esqueleto é um eixo simétrico que pode ser definido como o centro de todos os círculos de raio máximo inscritos dentro de uma forma. Uma forma pode ser descrita como o conjunto desses centros e

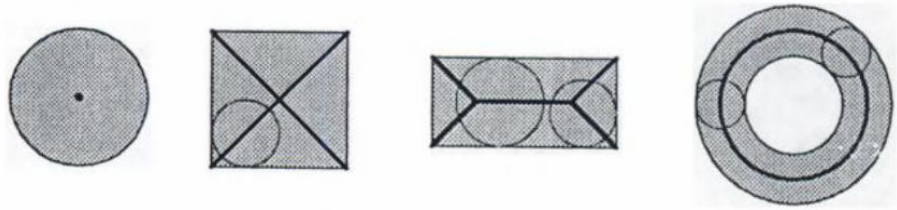


Figura 2.1 Exemplos de esqueletizações [Dou94]. O círculo é representado apenas por um ponto. O quadrado é representado por eixos que cruzam suas diagonais. As linhas pretas marcam os esqueletos do retângulo e da coroa.

os respectivos raios de cada centro. Este método pode tanto ser usado para compressão como para reconhecimento de formas. Exemplos de esqueletizações podem ser vistos na Figura 2.1. O esqueleto do círculo é apenas um ponto, pois ele é suficiente para tangenciar todos os pontos da borda da forma. Para o quadrado, círculos de raio menor são necessários para representar os pontos perto dos cantos.

Uma *quadtree* divide cada região da imagem em quatro regiões de tamanhos iguais, a região dividida se torna um nó e cada uma das quatro regiões formadas uma folha da *quadtree*. A decomposição por *quadtree* inicia a imagem como uma folha, então vai subdividindo cada folha até que a região representada por cada folha seja uma região homogênea, como pode ser visto na Figura 2.2.

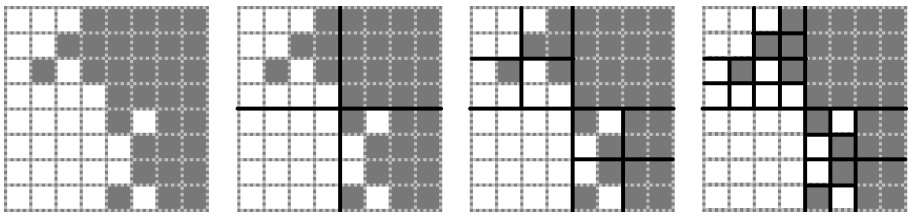


Figura 2.2 Exemplo de decomposição por quadtree. Cada região é dividida em quatro novas regiões até que todas as regiões se tornem uniformes.

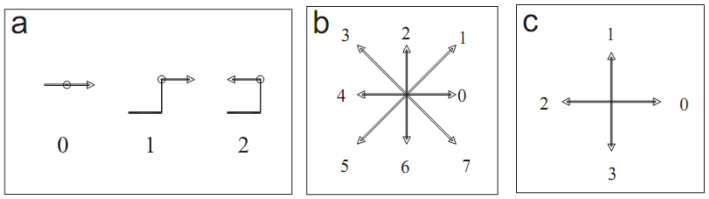


Figura 2.3 Exemplos de códigos conectividades usados em [SCBRD07]: (a) 3OT, direções ortogonais de cadeias de três símbolos (*orthogo- nal chain directions of three symbols*); (b) F8, cadeia de Freeman de oito direções; (c) F4, cadeia de Freeman de quatro direções.

2.1.3 Representação de Contorno

O método para codificação de contorno é a Codificação de Cadeia (*Chain Code*). Esta codificação interpreta o contorno de uma forma como uma cadeia de pixels. Para representar um contorno é necessário um ponto inicial. Cada ponto consecutivo da cadeia é representado através de um código que indica a posição em relação ao pixel atual. Cada ponto da cadeia é codificado até que se chegue ao ponto inicial ou ao fim de um contorno aberto. O código relativo atribuído a cada ponto pode seguir diversas conectividades, como pode ser visto na Figura 2.3. O exemplo de um contorno codificado pode ser visto na Figura 2.4, a conectividade usada para essa cadeia foi a F8 (cadeia de Freeman de oito direções), Figura 2.3 (b).

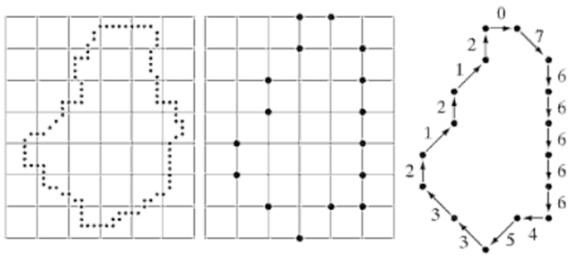


Figura 2.4 Exemplo de codificação de cadeia [GW10] de um contorno usando a conectividade F8 (cadeia de Freeman de oito direções) da Figura 2.3 (b).

2.2 Métodos de Compressão de Imagens Binárias

Os métodos de compressão revisados nesta seção têm a característica de serem próprios para a representação sem perdas de imagens binárias. Os algoritmos CCITT Group 3, CCITT Group 4 e JBIG foram criados especificamente para essa tarefa. Já os algoritmos GIF e PNG foram criados para representar imagens sem perdas não se restringindo a imagens binárias, mas esses algoritmos podem ser configurados para obterem uma maior taxa de compressão quando usados para representar imagens binárias. Algumas referências sobre esses métodos de compressão são [WMB99, Bar06].

2.2.1 CCITT Group3 e Group4

A pesquisa sobre compressão de imagens binárias digitais se iniciou com o desenvolvimento de padrões para transmissão de fax. No final dos anos 1970, o CCITT (*Comité Consultatif International Téléphonique et Télégraphiquere*) conheceu o potencial das tecnologias para transmissão de fax e passou a procurar novos padrões e soluções para o mesmo. Em 1980, o CCITT, através da recomendação T.4, normatizou o padrão de transmissão Group 3 [IT03] para a transmissão digital de fax para redes analógicas, que é até hoje um padrão de compressão de arquivo utilizado em máquinas de fax. Apesar de vários outros padrões de imagens binárias comprimidas apresentarem taxas de compressão maiores do que o Group 3, tais formatos são sensíveis a erros durante a transmissão fazendo com que o Group 3 ainda seja o padrão adotado para transmissão de fax via rede telefônica há 40 anos [Bar06].

Este padrão incorpora técnicas simples e rápidas de compressão de dados ao mesmo tempo em que realiza a detecção e correção de erros. O Group 3 substituiu os padrões analógicos Group 1 e Group 2. O Group 1 era não-confiável e era necessário em torno de seis minutos para transmitir cada página, enquanto o Group 2 transmitia a uma taxa de três minutos por página. O método de compressão usado no padrão Group 3 reduziu o tempo de transmissão de um fax para 6 a 30 segundos por página (mais 15 extras para a primeira página).

Em 1984, o CCITT publicou o padrão Group 4 [IT88], que é semelhante ao Group 3 mas

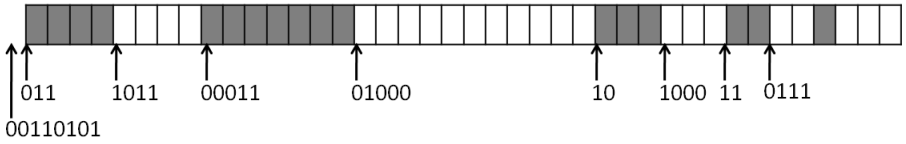
é destinado ao uso de redes de dados digitais em vez de circuito de telefone analógico convencional. Ele atinge aproximadamente o dobro da taxa de compressão do Group 3, no entanto, ele não realiza a detecção e correção de erros.

O padrão Group 3 se divide em dois métodos de codificação: unidimensional, chamado MH (*Modified Huffman*), e bidimensional, chamado MR (*Modified READ*, em que READ significa *Relative Element Address Designate codes*). O primeiro esquema trata cada linha independentemente enquanto o segundo utiliza MH para transmitir a primeira linha e transmite as outras linhas usando uma codificação relativa à linha anterior.

O MH codifica seqüências de bits de uma mesma cor através de um código. Este método é baseado na codificação Huffman [Huf52]. Ele associa códigos com menos bits a seqüências mais freqüentes. A Figura 2.5(b) mostra as doze primeiras linhas da tabela de códigos do MH. Essa tabela foi construída a partir de um conjunto de imagens de documentos selecionados pelo CCITT. Nota-se que a tabela de códigos diferencia seqüências de pixels pretos ou brancos.

A Figura 2.5(a) mostra os códigos gerados, empregando essa tabela, para uma linha de uma imagem binária. O MH assume que a primeira seqüência de pixels é branco e por isso indica que essa seqüência tem tamanho zero e é representada pelo código binário 00110101. Em seguida a seqüência de quatro pixels pretos é representada pelo código 011. O quatro pixels brancos seguintes são representados pelo código 1011. Os sete pixels pretos consecutivos são representados por 00011. O mesmo se dá para as demais cadeias de pixels da mesma cor.

Na codificação MR (Group 3 bidimensional), a cada k linhas, apenas a primeira é codificada pelo método unidimensional (MH). Todas as outras vão seguir o esquema onde a linha atual faz referência à anterior. Essa abordagem aumenta a compressão. É necessário transmitir uma linha via MH a cada k linhas para diminuir o erro de transmissão. O MMR é o algoritmo de compressão usado pelo padrão Group 4, seu funcionamento é igual ao MR. Porém, por se basear numa rede digital que já garante a integridade dos dados, apenas a primeira linha precisa ser transmitida via MH, as demais linhas são representadas em função da linha anterior.



(a)

White run length	Code word	Black run length	Code word
0	00110101	0	0000110111
1	000111	1	010
2	0111	2	11
3	1000	3	10
4	1011	4	011
5	1100	5	0011
6	1110	6	0010
7	1111	7	00011
8	10011	8	000101
9	10100	9	000100
10	00111	10	0000100
11	01000	11	0000101
...	...	...	...

(b)

Figura 2.5 Exemplo da codificação MH (*Modified Huffman*) empregada pelo padrão Group 3. A Figura (a) é uma linha de uma imagem binária codificada pelo algoritmo MH. Cada seqüência de pixels da mesma cor é representada por um código que indica seu tamanho. Cada código está definido na tabela do MH, as doze primeiras linhas dessa tabela estão exibidas na Figura (b). A Figura (a) inicia com um código indicando que o número de pixels brancos no começo da linha é zero pois o MH assume que o primeiro pixel de cada linha é branco.

2.2.2 GIF e PNG

GIF (*Graphics Interchange Format*) foi um dos formatos de compressão de imagem sem perda de qualidade mais usado até 1995. O padrão foi introduzido pela CompuServe em 1987 com o objetivo de ser um formato de imagens gráficas que podem ser exibidas em diferentes plataformas. Por isso é amplamente usado na Internet, tanto na forma de figuras fixas como animadas.

GIF codifica o arquivo de modo que cada pixel é representado por 8 bits ou menos. Cada código corresponde a uma cor de uma paleta limitada a 256 cores. Essa paleta pode ser especificada para cada imagem ou grupo de imagens que compartilham do mesmo esquema de cor. Imagens binárias utilizam uma paleta reduzida, com apenas duas cores. Cada cor é do tipo RGB e quando gravada no cabeçalho do arquivo consome 24 bits - 8 bits para cada cor primária: vermelho, verde e azul.

A compressão usada pelo formato é o LZW (Lempel-Ziv-Welch). Esta inicializa um dicionário com o alfabeto, cada instância do alfabeto é uma cor, em seguida analisa frases sucessivas da seqüência de entrada. Estas podem ser adicionadas ao dicionário. O esquema de codificação GIF inicia o dicionário com até 256 possíveis códigos para os pixels, mais dois códigos extras, o código "limpar"e "fim". Então processa a codificação LZW para a seqüência de pixels.

O LZW é uma das variações mais populares dos princípios de codificação Ziv-Lempel e foi baseado no algoritmo LZ78 [WMB99]. O método codifica apenas frases numéricas e não tem caracteres explícitos na saída. O algoritmo inicia a lista de frases (dicionário) com todos os caracteres da entrada. Uma nova frase é formada a partir de um elemento já existente anexado ao primeiro caractere da entrada atual.

A Figura 2.6 mostra o exemplo de uma execução do algoritmo LZW. A cadeia de entrada é abaababbaabaabaa. O dicionário é iniciado apenas com os símbolos do alfabeto a=1,b=2. O laço do algoritmo procura a maior cadeia presente no dicionário para codificar os símbolos da cadeia de entrada na posição atual da iteração. A cada trecho da cadeia de entrada codificado, um código é inserido na entrada de saída e uma nova instância é criada no dicionário. A nova instância do dicionário criada a cada iteração é a instância que foi codificada contatenada com

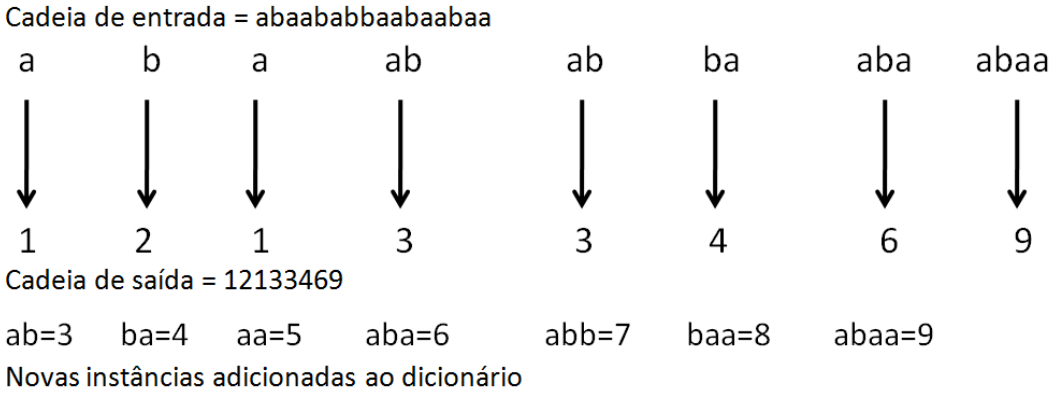


Figura 2.6 Exemplo da execução do algoritmo LZW. A cadeia de entrada (representada por letras) é convertida numa cadeia de índices para instâncias do dicionário. A cada índice inserido na cadeia de saída, uma nova instância é adicionada ao dicionário.

o próximo símbolo. No exemplo da Figura 2.6, o símbolo *a* é codificado pelo código 1 do dicionário, e a instância *ab* é inserida no dicionário com o código 3. Em seguida *b* é codificado na cadeia de saída com o código 2, e a instância *ba* é inserida no dicionário com o código 4. O código *a* é represento por 1 e a instância *aa* = 5 é inserida no dicionário. Continuando, algoritmo localiza na cadeia a instância *ab* do dicionário, que é codificada como 3 na cadeia de saída. O algoritmo repete esse laço até toda a cadeia estar codificada.

Com base no GIF foi gerado um novo método de compressão sem perda, o PNG (*Portable Network Graphics*). Este pretendia substituir o GIF para evitar infringir a patente de Unisys sobre a técnica LZW. O método PNG é similar ao GIF, ambos geram uma seqüência com os valores dos pixels e empregam o propósito geral de compressão Ziv-Lempel. O PNG também tem melhores taxas de compressão e mais recursos.

O padrão PNG usa o esquema de compressão gZip e incorpora elementos opcionais, como os "filtros", aplicados antes da compressão. Para cada linha de imagem um tipo de filtro é escolhido, este prevê o valor de cada pixel com base nos valores dos pixels vizinhos. PNG também adiciona uma série de outras melhorias em relação ao GIF. Ele não fica limitado a apenas 256 cores de 24 bits, também suporta paletas de 48 bits, tons de cinza (16 bits) e permite 256 níveis de transparências para cada pixel.

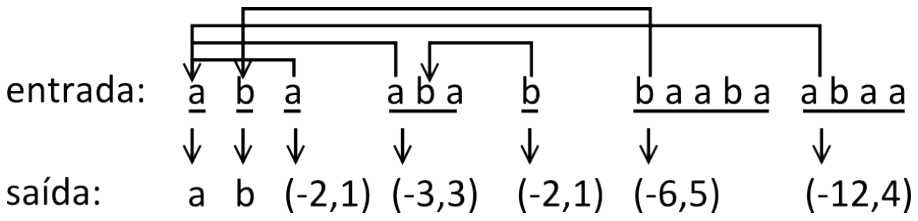


Figura 2.7 Exemplo do funcionamento do algoritmo gZip, uma série de caracteres é substituída por um ponteiro para a anterior, sob a forma de uma tupla (distância, comprimento).

O gZip é mais uma variação de compressão Ziv-Lempel e foi baseado no algoritmo LZ77 [ZL77]. O método utiliza o algoritmo de deflação [GA10, GA96] e a Codificação de Huffman para gerar e gravar seus resultados. O algoritmo de deflação encontra seqüências repetidas na entrada. A segunda ocorrência de uma série de caracteres é substituída por um ponteiro para a anterior, sob a forma de uma tupla (distância, comprimento), Figura 2.7. As distâncias são limitadas a 32 kbytes e representa a extensão do símbolo atual até o primeiro da seqüência que se repete, e os comprimentos são limitados a 258 bytes e significam o tamanho dessa seqüência. Se não for encontrada nenhuma recorrência nesse intervalo, é gerado um literal no lugar do ponteiro. As séries de símbolos duplicadas são encontradas utilizando uma tabela *hash*. Os literais e os comprimentos (da tupla) são compactados com a mesma árvore de Huffman, e as distâncias são compactadas com outra árvore. O arquivo de entrada é fragmentado em vários blocos, para cada bloco o algoritmo é aplicado e novas árvores de Huffman são criadas.

2.2.3 JBIG e JBIG2

JBIG, também conhecido como JBIG1, é um padrão internacional para compressão sem perda de imagens. É destinado a imagens binárias, mais especificamente para fax, porém pode ser usada para tons de cinza comprimindo cada plano de bits separadamente. A compressão pode ser aplicada à imagem inteira, ou pode usar o modo progressivo. No modo progressivo a compressão é aplicada, separadamente, a uma versão de menor resolução da imagem (chamada camada base) e a imagens que guardam a informação complementar para gerar versões em maiores resoluções da mesma imagem (chamadas camadas diferenciais). A codificação é baseada no uso

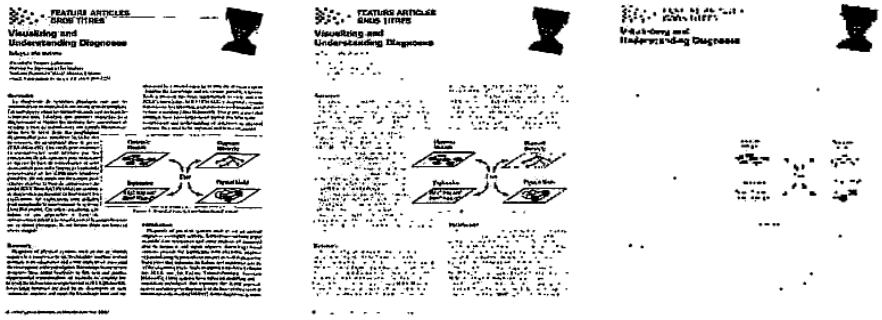


Figura 2.8 Exemplo de várias resoluções de uma mesma imagem. As duas imagens da esquerda apresentam maior resolução e formam as camadas diferenciais do JBIG. A imagem da direita apresenta menor resolução e forma a camada base do JBIG.

de *templates* e uma codificação aritmética adaptativa. Esta é baseada na probabilidade de cada bit em relação aos bits anteriores do *template*, permitindo uma compressão e descompressão em ordem de leitura.

A redução de resolução funciona de forma simples, para dividi-la pela metade, agrupam-se pixels em blocos de 2×2 e faz-se a média dos quatro tons de cada bloco. Porém isso só seria possível para imagens em tons de cinza, no caso de imagens binárias é necessário adaptar esse algoritmo. Uma solução fácil é usar uma função randômica em caso de empate (tantos blocos brancos quanto pretos), como a redução aplicada à imagem da Figura 2.8. Essa solução gera muito ruído na imagem, a melhor solução seria usar regras mais complexas para associar os pixels. Para cada bloco seria atribuído um valor calculado em função dos seus vizinhos (cor e posição), o pixel que substituirá os 4 blocos terá a cor em função desses valores.

Para o uso de *templates* o JBIG define dois tipos de contexto, uma para a base e outro para as camadas diferenciais. Para a camada base são usados dois *templates* diferentes como mostrado na Figura 2.9, ambos com 10 pixels. Para as camadas diferenciais há quatro *templates* diferentes, todos com 6 pixels e usados dependendo da fase de codificação do pixel. A idéia do *template* é servir de molde para calcular a codificação aritmética para um determinado pixel, CAE (*Context-based Arithmetic Encoding*). A probabilidade utilizada pela codificação é calculada para cada pixel em relação aos pixels do *template* (marcados pelas bolas pretas na Figura 2.9), isto é, a probabilidade é sensível ao contexto.

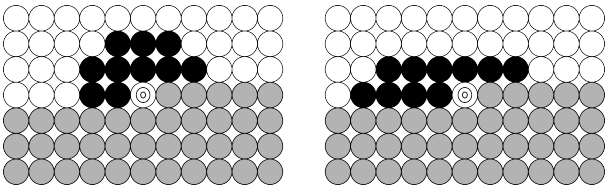


Figura 2.9 *Templates* da camada base. As bolas cinzas são os pixels a serem codificados, as pretas e brancas são pixels já codificados e o *alvo* é o pixel atual. As bolas pretas representam os pixels que fazem parte do *template*.

JBIG2 é o estado da arte em compressão de imagens binárias. O método é uma otimização do JBIG e faz compressão com ou sem perdas. O padrão foi planejado para ter um desempenho melhor que os métodos existentes na compressão sem perda, e na com perda para atingir taxas muito elevadas de compressão sem prejuízos significativos de qualidade visual. O JBIG2 é útil para aplicação em fax, armazenamento de documentos e arquivamento, codificação de imagens na Internet, transmissão de dados sem fio, fila de espera de impressão, e até mesmo teleconferência. O padrão JBIG2 define explicitamente a decodificação da compressão, ele não define explicitamente o padrão de codificação. Porém é suficientemente flexível para permitir que a concepção de um codificador sofisticado.

Outro diferencial do JBIG2 é a decomposição do documento de imagem em páginas, e essas em várias regiões. Cada uma dessas regiões é tratada de forma diferente, de acordo com seu tipo. Estas podem ser de símbolo, *halftone* ou genérica. As regiões de símbolo são áreas que contém textos, caracteres pequenos dispostos em linhas horizontais ou verticais. Esses caracteres são chamados de símbolos. O JBIG2 obtém grande parte de sua eficácia reutilizando símbolos individuais que se repetem. Estes são coletados em um ou mais dicionários. Estes indexam um conjunto de *bitmaps* de símbolos aos números de índice. Uma página pode conter dados em *halftone*, como imagens geradas por fotografias. Uma região de *halftone* consiste em um número de padrões distribuídos em uma grade regular. Essas regiões são codificadas com um dicionário de padrão de *halftone*, os padrões geralmente correspondem a valores de escala de cinza.

Uma página também pode conter outro tipo de dados, como linhas e ruídos, caracterizando a região genérica. Essas áreas podem ser codificadas com dois métodos diferentes, o primeiro

é conhecido como *Modified-Modified-Read* (MMR), e é usado pelo Grupo 4 para fax. O segundo método é uma variação do método usado pelo JBIG, faz uso de *templates* e codificação aritmética CAE (*Context-based Arithmetic Encoding*). Este método atinge maiores taxas de compressão e é mais eficaz para regiões com linhas, figuras e gráficos.

2.3 Reconhecimento de Forma

Descritores de forma são empregados para comparar silhuetas de objetos bidimensionais determinando uma medida de similaridade. Estes descritores são úteis e importantes para atividades como recuperação de imagens numa base de dados. Uma definição de forma retirada de [LR01] diz que a forma de um objeto é a característica que permanece invariante sob qualquer translação, rotação e reflexão do objeto. Espera-se, portanto, que descritores de forma apresentem tais características, e em alguns casos ainda espera-se invariância à escala.

Em informação multimídia, objetos bidimensionais geralmente são projeções de objetos tridimensionais. Este fato pode alterar a silhueta dos objetos de três maneiras: mudanças de ponto-de-vista; movimentos não rígidos do objeto (como uma bandeira flamulando, ou um cachorro correndo); ou inserção de ruído durante a digitalização. MPEG-7 é um padrão de interface para descrição de conteúdo multimídia (*Multimedia Content Description Interface*) amplamente utilizado e que incorpora descritores de forma [Mar04]. Para avaliar a qualidade dos descritores de forma sob estas três condições citadas, o MPEG-7 propôs um método de avaliação para os descritores sob um conjunto de base de dados chamado *Core Experiment CE-Shape-1* [LLE00], cada uma dessas bases é formada de imagens binárias de silhuetas de objetos.

A base de imagens *Core Experiment CE-Shape-1* é dividida em três partes: A, B e C. A parte A é subdividida em A1, que avalia a robustez do descritor a variações de escala, e A2, que avalia a robustez do descritor a variações de rotação. A parte B avalia a performance do descritor em relação à recuperação de imagens similares. E a parte C avalia a robustez do descritor em relação a movimentos não rígidos.

A base de imagens A1, que avalia a robustez do descritor a variações de escala, contém 420 formas. São 70 formas apresentadas em seis escalas diferentes cada. As escalas são: $2\times$, $1\times$, $0{,}3\times$, $0{,}25\times$, $0{,}2\times$ e $0{,}1\times$. Exemplos de imagens da base A1 podem ser visualizados na Figura 2.10. O método de teste padrão para a base A1 é recuperar, para cada uma das 420 imagens, as 6 imagens mais similares, através de uma função de similaridade que utiliza o descritor de cada imagem. Conta-se um acerto cada vez que uma das seis imagens da mesma classe é recuperada, portanto o número máximo de acertos é $6 \times 420 = 2520$. E o acerto percentual é

$$100 \times \textit{número de acertos}/2520.$$

A base de imagens A2, que avalia a robustez do descritor a variações de rotação, contém 420 formas. São 70 formas apresentas em seis rotações diferentes cada. As rotações são: 0^o, 9^o, 36^o, 45^o (uma rotação de 9^o seguida de outra rotação de 36^o), 90^o e 150^o. Exemplos de imagens da base A2 podem ser visualizados na Figura 2.10. O método de teste padrão para a base A2 é recuperar, para cada uma das 420 imagens, as 6 imagens mais similares, através de uma função de similaridade que utiliza o descritor de cada imagem. Conta-se um acerto cada vez que uma das seis imagem da mesma classe é recuperada, portanto o número máximo de acertos é $6 \times 420 = 2520$. E o acerto percentual é

$$100 \times \textit{número de acertos}/2520.$$

A base de imagens B, que avalia a performance do descritor em relação à recuperação de imagens similares, é a parte mais importante do CE-Shape-1. Esta base possui 1400 imagens, sendo 20 imagens por classe, algumas dessa imagens podem ser vistas na Figura 2.11. O teste para a base B utiliza cada uma das 1400 imagens, compara com todas, incluindo ela mesma, e recupera as 40 imagens mais similares. O número máximo de imagens corretas recuperadas corretamente é 20 por classe, então o número máximo de acertos é 28000. E o acerto percentual é

$$100 \times \textit{número de acertos}/28000.$$

A base de imagens C, que avalia a robustez do descritor em relação a movimentos não rígidos, é composta de 1300 imagens, sendo 200 quadros de um vídeo de um peixe nadando e

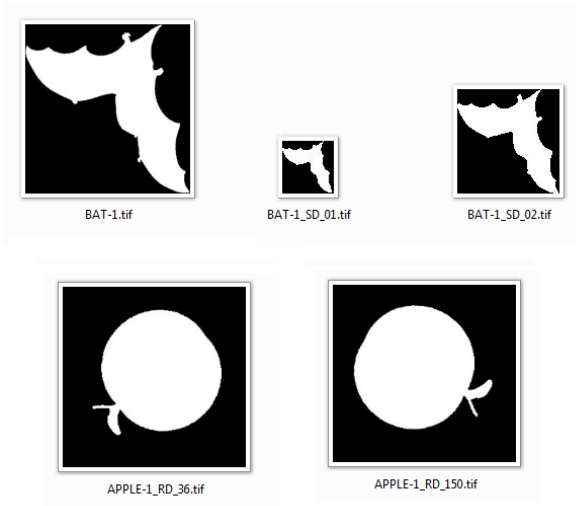


Figura 2.10 Exemplos da base de imagens binárias de silhuetas de objetos *Core Experiment CE-Shape-1* parte A. A linha de cima contém imagens da base A1, que contém em cada classe uma mesma imagem em escalas diferentes. A linha de baixo contém imagens da base A2, que contém, por classe, uma mesma imagem em diferentes rotações.

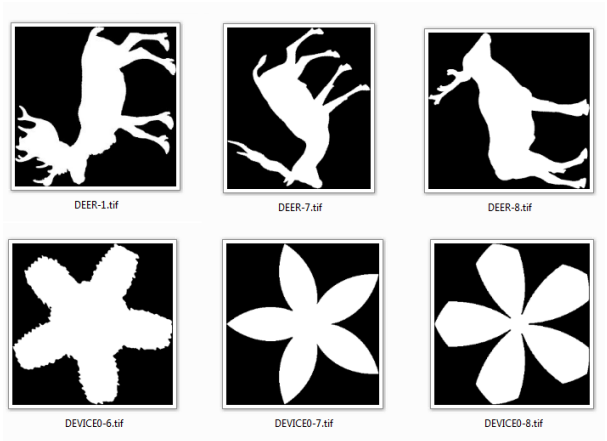


Figura 2.11 Exemplos da base de imagens binárias silhuetas de objetos *Core Experiment CE-Shape-1* parte B. A linha de cima contém 3 das 20 imagens da classe DEER. E a linha de baixo contém 3 das 20 imagens da classe DEVICE. Percebe-se que os elementos de cada classe apesar de semelhantes representam formas de objetos distintos.

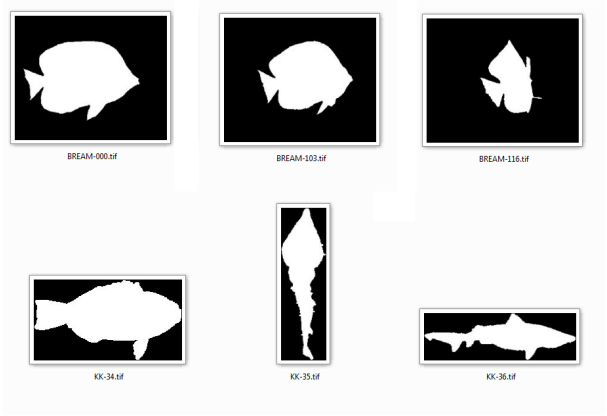


Figura 2.12 Exemplos da base de imagens binárias silhuetas de objetos *Core Experiment CE-Shape-1* parte C. A linha de cima da figura contém quadros do vídeo de um peixe nadando, a imagem mais à esquerda da primeira linha primeira dos 200 quadros do vídeo. A linha de baixo contém exemplos das 1100 imagens de outros peixes e animais marinhos.

1100 imagens de outros peixes e animais marinhos. Exemplos de imagens da base C podem ser visualizados na Figura 2.12. O método de teste para a base C é recuperar as 200 imagens mais próximas da imagem que forma o primeiro dos 200 quadros do vídeo. Portanto a taxa de acerto percentual é medida como

$$100 \times \textit{número de acertos}/200.$$

Alguns dos descritores de forma incorporados pelo padrão MPEG-7 estão listados a seguir:
P298 utiliza o descritor de forma de espaço tangencial baseado na melhor correspondência de partes visuais [LL00]. Um único contorno é usado como um descritor de forma, este contorno é determinado de forma ótima através de um processo de simplificação do contorno chamado evolução discreta de curva;
P320 utiliza o descritor de forma CSS (*Curvature Scale Space*). Um contorno simplificado é obtido através da sua suavização através da convolução com funções gaussianas [dA07];
P517 utiliza autovetores de múltiplas camadas. Os autovetores são calculados para várias regiões da imagem e são usados para formar o descritor [Zib01];
P567 utiliza um descritor de forma wavelet onde o contorno é representado em várias res-

oluções através das funções de aproximação e funções de detalhes wavelets [CK96];
P687 utiliza momentos de Zernike, que além de terem um embasamento teórico e experimental bem estabelecidos, apresenta melhores resultados do que momentos convencionais e momentos invariantes [KH90].

Os resultados dos testes na base *Core Experiment CE-Shape-1* de cada um dos descritores de forma citados anteriormente está descrito na Tabela 2.1. Os descritores de forma P298 e P320 apresentam melhores resultados do que os outros. Ambos os descritores utilizam versões simplificadas dos contornos, obtidas através de processos de evolução de curvas distintos, para a comparação de formas. Quando as bases do *CE-Shape-1* foram testadas com seres humanos, isto é, o classificador das formas é uma pessoa, a taxa máxima de acerto para as bases A1 e C foi de 93% em cada, a base B também não apresentou 100% de acerto [LLE00].

Tabela 2.1 Taxas de acerto percentual dos descritores P298, P320, P517, P567 e P687sobre as bases *Core Experiment CE-Shape-1* A1, A2, B e C.

Base	P298	P320	P517	P567	P687
A1	88,65	89,76	92,42	88,04	92,64
A2	100,0	99,37	100,0	97,46	99,60
B	76,45	75,44	70,33	67,76	70,22
C	92,00	96,00	88,00	93,00	94,50

CAPÍTULO 3

Código de Vizinhança

É proposta em [TTD99] uma nova maneira de codificar imagens através de relações de vizinhança. A codificação proposta transforma cada pixel preto de uma imagem binária em um vetor de vizinhança. Este vetor possui quatro dimensões. A primeira posição guarda o número de pixels do pixel preto que está sendo codificado até o próximo pixel branco acima ou no fim (na borda) da imagem na mesma direção, por isso essa posição do vetor é chamada Norte. A segunda posição é chamada Leste e é medida analogamente à primeira contudo noutra direção, à direita. As posições seguintes são Sul e Oeste e são medidas da mesma maneira nas direções abaixo e à esquerda, respectivamente. Portanto um vetor de vizinhança pode ser escrito como $V = (n,l,s,o)$. A Figura 3.1 mostra um componente conexo de uma imagem binária, os quadrados cinza destacam a vizinhança representada pelo vetor de vizinhaça V = (5,2,9,4) e os quadrados brancos representam os demais pixels do componente conexo. Em [TTD99], esse vetor é então transformado em um código, que é um número inteiro. O método usado para fazer essa transformação gera um número muito grande de possíveis códigos e, como um código é gerado para cada pixel preto da imagem, muitos códigos distintos são produzidos. Portanto os vetores de vizinhança são re-escalados para que o número de possíveis códigos seja no máximo 100. Cada imagem é transformada num vetor de 100 características onde cada característica é a freqüência de cada um dos possíveis códigos. Uma rede neural é treinada com esses vetores de características e aplicada, com sucesso, para o problema de reconhecimento de caracteres manuscritos.

Em [TT06a, TT06b] outras abordagens são utilizadas para o reconhecimento de padrões empregando os códigos extraídos a partir dos vetores de vizinhança, além disso são propostos os operadores de vizinhaça, que são funções lógicas e matemáticas aplicadas aos vetores de vizinhança que apresentam comportamento semelhantes a operadores morfológicos em imagens binárias. A análise de chi-quadrado é usada em [TT06a] para o reconhecimento de caracteres

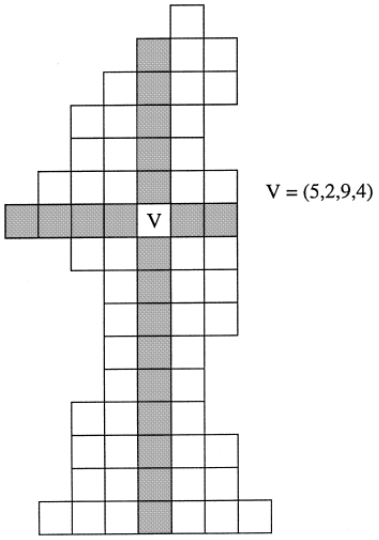


Figura 3.1 Vetor de vizinhança proposto por [TTD99]. A grade é conjunto de pixels pretos da imagem. Os quadrados cinza são os pixels representados pelo vetor de vizinhança V = (5,2,9,4) – 5 pixels na direção direção Norte, 2 na direção Leste, 9 na direção Sul e 4 na direção Oeste, $V = (n, l, s, o)$.

manuscritos. Em [TT06b] os métodos de agrumamento k-Média e Fuzzy k-Médias são usados para reduzir o número de códigos e novamente empregrados na tarefa de reconhecimento de caracteres manuscritos através de redes neurais.

Uma grande limitação dos vetores de vizinhança descritos é o fato de não ser possível reconstruir a imagem original, como é discutido em [TTD99]. O presente trabalho propõe uma nova definição para os vetores de vizinhaça chamada código de vizinhança. Usando essa nova codificação proposta é possível reconstuir a imagem sem perdas. A nova definição de código de vizinhança também permite a definição de novas vizinhanças, que são chamadas braços. Ainda são propostos três algoritmos para a redução do número de códigos de vizinhança, sendo que um deles utiliza uma abordagem evolucionária.

3.1 Imagem Binária como Matriz de Bits

Uma **imagem binária** é uma **matriz binária** $P = [p_{xy}]$, de dimensões $w \times h$, em que cada posição $(x, y) \in \mathbb{N} \times \mathbb{N}$ da matriz só pode assumir dois valores, 0 ou 1, $p_{xy} \in \{0, 1\}$. Quando

o valor p_{xy} de uma posição (x,y) da matriz é 0 também é chamado **branco**, pixel branco ou pixel de fundo, quando esse valor é 1 é chamado **preto**, pixel preto ou pixel da imagem.

O número de colunas da matriz P é w e também pode ser chamado largura da imagem e h, número de linhas, é chamado altura da imagem. Diferentemente da notação convencional de matrizes [Kol98] onde a primeira posição é referente às linhas e a segunda às colunas, a notação usada aqui foi modificada não só por questões de conveniência mas também para ficar conforme a notação comumente empregada na área da processamento de imagens [GW10]. Acredita-se que essa pequena alteração de notação não compromete a generalidade da representação.

Pode-se então definir uma imagem binária, semelhantemente a uma matriz, da seguinte forma: uma imagem binária P de dimensões $w \times h$ é um arranjo retangular de wh números binários arrumado em w colunas verticais e h linhas horizontais:

$$P = \begin{bmatrix} p_{00} & p_{10} & p_{20} & \cdots & p_{(w-1)0} \\ p_{01} & p_{11} & p_{21} & \cdots & p_{(w-1)1} \\ p_{02} & p_{12} & p_{22} & \cdots & p_{(w-1)2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{0(h-1)} & p_{1(h-1)} & p_{2(h-1)} & \cdots & p_{(w-1)(h-1)} \end{bmatrix}. \tag{3.1}$$

A x-ésima coluna de P é

$$\begin{bmatrix} p_{x0} \\ p_{x1} \\ p_{x2} \\ \vdots \\ p_{x(h-1)} \end{bmatrix} \quad (0 \leqslant x < w); \tag{3.2}$$

a y-ésima linha de P é

$$\begin{bmatrix} p_{0y} & p_{1y} & p_{2y} & \cdots & p_{(w-1)y} \end{bmatrix} \quad (0 \leqslant y < h). \tag{3.3}$$

Mais uma vez a notação usada aqui difere, ainda sem perda de generalidade, daquela padrão de matrizes. Em vez de usar (i,j) para representar o **elemento** ou **componente** da linha i e coluna j, usa-se (x,y) para localizar p_{xy}, o **pixel** da coluna x e da linha y.

Uma interpretação gráfica diz que a posição $(0,0)$ da matriz representa o canto superior esquerdo da imagem, enquanto a posição $(w-1,h-1)$ representa o canto inferior direito da imagem. $(w-1,0)$ é a posição do pixel mais a direita da primeira linha da imagem (canto superior direito) e $(0,h-1)$ é a posição do primeiro pixel da última linha da imagem (canto inferior esquerdo).

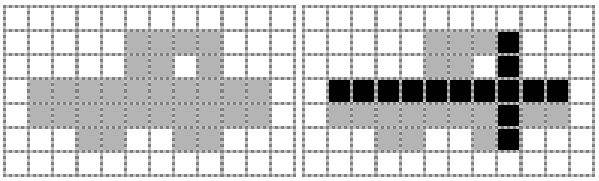
3.2 Codificação de Vizinhança

A codificação de vizinhança é uma representação de imagens binárias que transforma uma matriz de bits em um conjunto de códigos de vizinhaça. A partir desse conjunto de códigos é possível reconstruir completamente a imagem binária representada por ele, ou seja, é possível reconstruir uma matriz de bits equivalente àquela usada para gerar os códigos de vizinhança.

XC[1] é a sigla para designar um código de vizinhança. Cada XC é um vetor de números inteiros que representa um conjunto de pixels de uma imagem binária (um conjunto de posições na matriz de bits). Esse vetor é constituído de duas partes: a posição do centro do código dentro da imagem e a vizinhança ou braços deste centro. O centro do código é a posição do pixel ao redor do qual será codificada a vizinhança, o centro é representado pelo par ordenado (x,y), com $x,y \in \mathbb{N}$. Cada posição da parte de vizinhança é chamada comprimento do braço, esta parte é semelhante ao vetor de vizinhança proposto em [TTD99], com duas diferenças principais: o número de vizinhanças é variável em vez de exatamente quatro como proposto no artigo; é possível usar outros tipos de vizinhança além de norte, leste, sul e oeste.

A Figura 3.2(b) destaca um XC da imagem binária da Figura 3.2(a). Esse XC destacado é $(x=8,y=3,n=2,l=2,s=2,o=7)$ e possui centro na posição $(8,3)$. A parte dos braços é um vetor de vizinhança semelhante ao proposto em [TTD99] que pode ser escrito, usando a notação da referência, como $(2,2,2,7)$, ou seja, dois pixels ao norte, dois ao leste, dois ao sul e sete ao oeste.

[1] A sigla XC foi escolhida pelo autor da seguinte maneira: a letra X apresenta certa semelhança gráfica com os códigos de vizinhança originais [TTD99] e a letra C faz referência à palavra código.



(a) (b)

Figura 3.2 Exemplo de um código de vizinhança numa imagem binária. (a) Exemplo de uma imagem binária de dimensões 12×7, as células cinza da grade representam os pixels pretos e as demais células representam os pixels brancos; (b) É destacado um código de vizinhanças da imagem (a), este XC é descrito por $(x = 8, y = 3, n = 2, l = 2, s = 2, o = 7)$.

A codificação de vizinhança gera um vetor XC para cada pixel preto da imagem. Na Figura 3.3 é destacada a codificação de vizinhança de uma imagem 5×7 (largura 5 e altura 7). Esta imagem possui nove pixels pretos. Para cada um desses pixels é gerado um XC, o conjunto dos nove XCs é a codificação de vizinhança dessa imagem. As duas primeira posições do vetor representam a posição do centro (x, y) da imagem. As outra quatros posições representam a vizinhança de cada XC, no caso desse exemplo esta vizinhança é idêntica àquela (n, l, s, o) proposta em [TTD99].

3.3 Funções Braços – Vizinhanças

O código de vizinhança XC é um vetor de duas partes, a segunda parte de um XC corresponde à vizinhança de um centro, o qual é descrito na primeira parte deste vetor. Cada posição do vetor XC na parte de vizinhança é chamada braço ou comprimento do braço. Cada braço codifica um trecho da vizinhança. Fazendo uso desta nomeclatura pode-se dizer que o vetor de vizinhança utilizado em [TTD99] tem quatro braços, são eles: norte, leste, sul e oeste.

O presente trabalho cria o conceito de braços para que a parte de vizinhança do código XC seja versátil. Assim, através da combinação de definições de braços pode-se criar diversas configurações de XC. Uma configuração de um XC é uma especificação de quais funções braços o código usa para representar sua vizinhaça. A configuração empregada em [TTD99] usa as

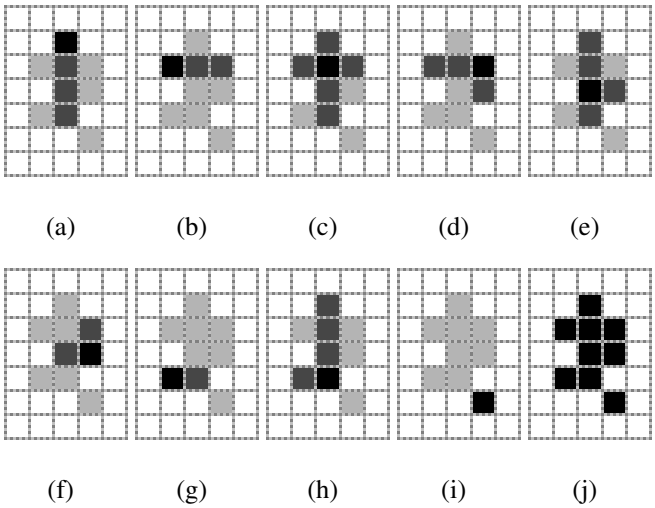


Figura 3.3 Codificação de Vizinhança. As Figuras de (a) a (i) destacam os 9 códigos de vizinhança da imagem (j), cada código é representado por um vetor da forma (x, y, n, l, s, o). O centro de cada código está destacado na imagem em preto e as vizinhanças em cinza escuro. Um XC é gerado para cada pixel preto da imagem. Códigos das imagens: a = (2,1,0,0,3,0, b = (1,2,0,2,0,0), c = (2,2,1,1,2,1), d = (3,2,0,0,1,2), e = (2,3,2,1,1,0), f = (3,3,1,0,0,1), g = (1,4,0,1,0,0), h = (2,4,3,0,0,1), i = (3,5,0,0,0,0).

funções braço norte, leste, sul e oeste. Outra configuração poderia usar somente as funções norte e leste ou ainda fazer uso apenas a função sul. Uma discussão mais aprofundada sobre configurações de XC é realizada nas próximas seções. Nesta seção, são definidas as funções braço, que são os elementos de uma configuração de XC.

Um braço pode ser definido como uma função b no domínio das triplas de números naturais $\mathbb{N}^3$ e contradomínio no conjunto das partes dos pares ordenados de números naturais menos o conjunto vazio $\wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing$:

$$b : \mathbb{N}^3 \to \wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing. \tag{3.4}$$

Esta função b pode ser interpretada com uma função que tem como parâmetros três números naturais – as coordenadas (x, y) do centro do código e o comprimento do braço – e retorna um conjunto de pontos da imagem binária, onde cada ponto é uma posição na matriz da imagem binária.

A posição (x, y) de qualquer pixel p_{xy} da imagem pode ser descrita como um elemento do conjunto dos pares ordenados de números naturais ou, equivalentemente, $(x, y) \in \mathbb{N} \times \mathbb{N}$.

O conjunto das partes dos pares ordenados de números naturais $\wp(\mathbb{N} \times \mathbb{N})$ representa todos os possíveis conjuntos formados pelos elementos de $\mathbb{N} \times \mathbb{N}$ incluindo o conjunto vazio $\varnothing$. O conjunto das partes dos pares ordenados de números naturais menos o conjunto vazio $\wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing$ contém todos os conjuntos de pares ordenados com pelo menos um par. Logo um elemento Π de $\wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing$ pode ser interpretado como um conjunto de posições de pixels da imagem [2],

$$\Pi = \{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\} \in \wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing. \tag{3.5}$$

Uma vez que cada pixel p_{xy} de uma imagem binária só pode assumir um entre dois valores, $p_{xy} \in \{0, 1\}$, é possível representar sem perdas uma imagem binária através das suas dimensões $w \times h$ e um conjunto Π com todas as posições (x, y) dos pixels pretos. Pois todas as posições (x, y) da imagem $P = [p_{xy}]$ que estão em Π representam pixels pretos ($p_{xy} = 1$) e as demais posições da matriz representam pixels brancos ($p_{xy} = 0$):

$$p_{xy} = \begin{cases} 1, \ se \ (x, y) \in \Pi \\ 0, \ se \ (x, y) \notin \Pi \end{cases}$$

A função braço b (eq. 3.4) codifica uma região da imagem gerando um conjunto $\Pi' \subseteq \Pi$ de posições (x, y) de pixels pretos desta imagem. Esse conjunto Π' é gerado a partir da posição do centro do código de vizinhança e do comprimento do braço:

$$b(x, y, c) = \{(x_1, y_1), (x_2, y_2), \ldots, (x_a, y_a)\} \ \in \ \wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing$$

Diversas funções braços b podem ser definidas a partir da equação 3.4. O conjunto de todas as funções braços b é chamado B, B é a letra grega beta maiúscula:

$$\mathrm{B} = \{b | b : \mathbb{N}^3 \to \wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing\}. \tag{3.6}$$

Nas subseções seguintes são definidas oito funções braços $b \in \mathrm{B}$. As quatro primeiras são chamadas funções braços horizontais e verticais, estas funções foram inspiradas no vetor de vizinhança proposto em [TTD99], são elas: norte, leste, sul e oeste. As quatro funções braço restantes são chamadas funções braço diagonais e apresentam certa semelhança com as primeiras, são elas: nordeste, sudeste, sudoeste e noroeste.

[2] Assume-se que uma imagem binária tenha ao menos um pixel preto.

3.3.1 Funções Braços Horizontais e Verticais

Os braços horizontais e verticais descritos nesta subseção são exatamente as vizinhanças proposta em [TTD99], contudo, usando a notação de funções braços da equação 3.4. Essas funções braços fazem analogia aos pontos cardeais, são elas: norte, leste, sul e oeste.

- **Norte (b_N)**

  A função braço norte b_N é uma função braço (equação 3.4) que gera um conjunto de pontos acima e na mesma coluna do centro (x,y). Portanto a função norte $b_N \in \mathrm{B}$ (eq. 3.6) pode ser definida como

  $$b_N(x,y,c) = \{(x,y-k) \mid k = 0,1,2,\ldots,c; k \leqslant y\}, \tag{3.7}$$

  em que x é a posição horizontal e y a posição vertical do centro (x,y), c é o comprimento do braço.

  Exemplos:

  $b_N(33,12,0) = \{(33,12)\}$,

  $b_N(0,12,3) = \{(0,12),(0,11),(0,10),(0,9)\}$,

  $b_N(0,2,3) = \{(0,2)(0,1),(0,0)\}$.

- **Leste (b_L)**

  A função braço leste é uma função braço (equação 3.4) que gera um conjunto de pontos à direita e na mesma linha do centro (x,y). Portanto a função leste (b_L) pode ser definida como

  $$b_L(x,y,c) = \{(x+k,y) \mid k = 0,1,2,\ldots,c\}, \tag{3.8}$$

  em que x é a posição horizontal, y a posição vertical do centro (x,y) e c é o comprimento do braço.

  Exemplos:

  $b_L(33,12,0) = \{(33,12)\}$,

  $b_L(12,0,3) = \{(12,0),(13,0),(14,0),(15,0)\}$.

- **Sul (b_S)**

  A função braço sul é uma função braço (equação 3.4) que gera um conjunto de pontos abaixo e na mesma coluna do centro (x,y). Portanto a função sul (b_S) pode ser definida como

  $$b_S(x,y,c) = \{(x,y+k) \mid k = 0,1,2,\ldots,c\}, \tag{3.9}$$

  em que x é a posição horizontal, y a posição vertical do centro (x,y) e c é o comprimento do braço.

  Exemplos:

  $b_S(33,12,0) = \{(33,12)\}$,

  $b_S(0,12,3) = \{(0,12),(0,13),(0,14),(0,15)\}$.

- **Oeste (b_O)**

  A função braço oeste é uma função braço (equação 3.4) que gera um conjunto de pontos à esquerda e na mesma linha do centro (x,y). Portanto a função oeste (b_O) pode ser definida como

  $$b_O(x,y,c) = \{(x-k,y) \mid k = 0,1,2,\ldots,c; k \leqslant x\}, \tag{3.10}$$

  em que x é a posição horizontal, y a posição vertical do centro (x,y) e c é o comprimento do braço.

  Exemplos:

  $b_O(33,12,0) = \{(33,12)\}$,

  $b_O(12,5,3) = \{(12,5),(11,5),(10,5),(9,5)\}$,

  $b_O(2,0,4) = \{(2,0),(1,0),(0,0)\}$.

3.3.2 Funções Braços Diagonais

Fazendo a mesma analogia aos pontos cardeais que as funções horizontais e verticais, as funções diagonais são chamadas: nordeste, sudeste, sudoeste e noroeste.

- **Nordeste (b_{NE})**

  A função braço nordeste é uma função braço (equação 3.4) que gera um conjunto de pontos na diagonal, acima e à direita, do centro (x,y). Portanto a função nordeste (b_{NE}) pode ser definida como

  $$b_{NE}(x,y,c) = \{(x+k, y-k) \mid k = 0,1,2,\ldots,c; k \leqslant y\}, \tag{3.11}$$

  em que x é a posição horizontal, y a posição vertical do centro (x,y) e c é o comprimento do braço.

  Exemplos:

  $b_{NE}(33,12,0) = \{(33,12)\}$,

  $b_{NE}(12,12,3) = \{(12,12),(13,11),(14,10),(15,9)\}$,

  $b_{NE}(0,2,4) = \{(0,2)(1,1),(2,0)\}$.

- **Sudeste (b_{SE})**

  A função braço sudeste é uma função braço (equação 3.4) que gera um conjunto de pontos na diagonal, abaixo e à direita, do centro (x,y). Portanto a função sudeste (b_{SE}) pode ser definida como

  $$b_{SE}(x,y,c) = \{(x+k, y+k) \mid k = 0,1,2,\ldots,c\}, \tag{3.12}$$

  em que x é a posição horizontal, y a posição vertical do centro (x,y) e c é o comprimento do braço.

  Exemplos:

  $b_{SE}(33,12,0) = \{(33,12)\}$,

  $bSE(0,12,3) = \{(0,12),(1,13),(2,14),(3,15)\}$.

- **Sudoeste (b_{SO})**

  A função braço sudoeste é uma função braço (equação 3.4) que gera um conjunto de pontos na diagonal, abaixo e à esquerda, do centro (x,y). Portanto a função sudeste (b_{SO})

pode ser definida como

$$b_{SO}(x,y,c) = \{(x-k, y+k) \mid k = 0,1,2,\ldots,c; k \leqslant x\}, \tag{3.13}$$

em que x é a posição horizontal, y a posição vertical do centro (x,y) e c é o comprimento do braço.

Exemplos:

$b_{SO}(33,12,0) = \{(33,12)\}$,

$b_{SO}(12,12,3) = \{(12,12),(11,13),(10,14),(9,15)\}$,

$b_{SO}(1,12,3) = \{(1,12),(0,13)\}$.

- **Noroeste (b_{NO})**

  A função braço noroeste é uma função braço (equação 3.4) que gera um conjunto de pontos na diagonal, acima e à esquerda, do centro (x,y). Portanto a função sudeste (b_{NO}) pode ser definida como

$$b_{NO}(x,y,c) = \{(x-k, y-k) \mid k = 0,1,2,\ldots,c; k \leqslant min[x,y]\}, \tag{3.14}$$

  em que x é a posição horizontal, y a posição vertical do centro (x,y) e c é o comprimento do braço.

  Exemplos:

  $b_{NO}(33,12,0) = \{(33,12)\}$,

  $b_{NO}(12,12,3) = \{(12,12),(11,11),(10,10),(9,9)\}$,

  $b_{NO}(1,12,5) = \{(1,12),(0,11)\}$,

  $b_{NO}(12,1,4) = \{(12,1),(11,0)\}$.

3.4 Código de Vizinhança (XC)

O código de vizinhança (XC) é um vetor que possui duas partes: o centro do código na imagem e os comprimentos dos braços. Nas duas primeiras posições do vetor está a posição (x,y)

do centro do código e nas posições restantes os comprimentos dos braços que formam sua vizinhança. O centro do código não é o centro geométrico do XC nem o centro de massa dos pontos codificados pelo XC, é apenas um ponto da imagem que serve de referência para uma vizinhança.

Pode-se definir um **código de vizinhança** ou **vetor XC** φ como um vetor d-dimensional no espaço $\mathbb{N}^d$ com $d > 2$.

$$\varphi = (x, y, c_1, c_2, \ldots, c_{d-2}) \in \mathbb{N}^d,\ d > 2. \tag{3.15}$$

A primeira posição do vetor é chamada x e representa a coluna do centro do XC; a segunda posição, y, representa a linha do centro; as demais posições c_i, $1 \leqslant i \leqslant d-2$, são os comprimentos dos braços do XC, cada c_i está associado a exatamente uma função braço b_i (3.4). Portanto, para definir um XC é preciso definir quais funções braço estão associadas com cada comprimento.

Chama-se **configuração do XC** (ou tipo do XC) β é um vetor de funções braço b_i (3.4 que especifica quais funções braço são empregadas para definir a vizinhança de um código XC φ:

$$\beta = (b_1, b_2, \ldots, b_{d-2}) \in \mathrm{B}^{d-2},\ d > 2, \tag{3.16}$$

em que B é o conjunto de todas as funções braços (equação 3.6), $b_i \in \mathrm{B}$, $1 \leqslant i \leqslant d-2$ e d é a dimensão do vetor XC φ que utiliza a configuração β. A restrição $d > 2$ sobre a dimensão do vetor φ indica que *beta* tem, pelo menos, uma função braço e φ pelo menos um comprimento de braço.

O conjunto de todos os pontos gerados por um código XC φ pode ser recuperado através das funções braço b_i de sua configuração β e os respectivos comprimentos de braços c_i. A função $u : \mathbb{N}^d \times \mathrm{B}^{d-2} \to \wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing$ é chamada **função vizinhança de um XC**. Esta função associa um vetor XC φ e sua configuração β de funções braço a um conjunto de pontos numa imagem binária:

$$u(\varphi, \beta) = \bigcup_{i=1}^{d-2} b_i(x, y, c_i), \tag{3.17}$$

em que φ está definido na equação 3.15, β está definido na equação 3.16, b_i é cada função braço do vetor β, x é a posição horizontal do centro de φ, y é a posição vertical do centro de φ, c_i é o comprimento de cada braço de φ associado à função b_i e d é a dimensão de φ.

3.5 Configurações de XC

Nas Subseções 3.3.1 e 3.3.2 são definidas oito funções braço – b_L, b_N, b_{NO}, b_{NE}, b_O, b_S, b_{SE} e b_{SO} – que são empregadas nesta seção para construir configurações de XC β (eq. 3.16). O conjunto Θ dessas funções é um subconjunto do conjunto de todas as funções braço B (eq. 3.6):

$$\Theta = \{b_L, b_N, b_{NO}, b_{NE}, b_O, b_S, b_{SE}, b_{SO}\} \subset \mathrm{B}. \tag{3.18}$$

As funções braço de Θ apresentam uma propriedade em comum:

$$b_i(x,y,c_a) \subset b_i(x,y,c_b), c_a < c_b, bi \in \Theta, \tag{3.19}$$

isto é, o conjunto de pontos gerados por $b_i(x,y,c_a)$ é um subconjunto do pontos gerados por $b_i(x,y,c_b)$ quando $c_a < c_b$ para uma mesma função braço $b_i \in \Theta$, onde c_a e c_b são comprimento de braços e (x,y) é o centro do código. Esta propriedade afirma que, para uma mesma função braço $b_i \in \Theta$, os pontos gerados para um comprimento de braço $c_a < c_b$ também são gerados para o comprimento de braço $c_b > c_a$.

Uma conseqüência dessa propriedade é que, se uma função braço $b_i \in \Theta$ se repete numa configuração $\beta = (\ldots, b_i, \ldots, b_i, \ldots)$, não há ganho de representação de pontos através da função vizinhança u (eq. 3.17), pois

$$b_i(x,y,c_a) \cup b_i(x,y,c_b) \equiv b_i(x,y,max(c_a,c_b)), b_i \in \Theta$$

em que c_a e c_b são comprimentos de braços e (x,y) é o centro do código. Ou seja, para uma mesma função braço $b_i \in \Theta$ e comprimentos de braços c_a e c_b, a união do conjunto de pontos gerados por b_i para cada comprimentos de braços c_a e c_b é o mesmo conjunto gerado por b_i para o maior desses dois braços $max(c_a,c_b)$.

Portanto para uma configuração $\beta \in \Theta^d$ com $e < d$ funções braços repetidas no vetor β existe uma configuração $\beta' \in \Theta^{d-e}$ equivalemte, por exemplo,

$$(b_L, b_N, b_O, b_N, b_O) \equiv (b_L, b_N, b_O),$$

ou seja, a repetição das funções braço b_N e b_O não apresenta qualquer ganho de representação de pontos através da função vizinhança u (eq. 3.17), a configuração (b_L, b_N, b_O) consegue representar os mesmos pontos que a configuração $(b_L, b_N, b_O, b_N, b_O)$.

Outra característica das configurações de XC é a independência da posição das suas funções braço $b_i \in \mathrm{B}$ no vetor β. O importante é a associação entre o comprimento c_i do braço e sua respectiva função b_i para gerar os pontos da imagem através da função vizinhança u (eq. 3.17). Exemplo:

$$u((x,y,c_i,c_i),(b_i,b_i)) \equiv u((x,y,c_j,c_i),(b_j,b_i)),$$

em que (x,y) é o centro do código, c_i e c_j são os comprimentos dos braços e b_i e b_j suas respectivas configurações.

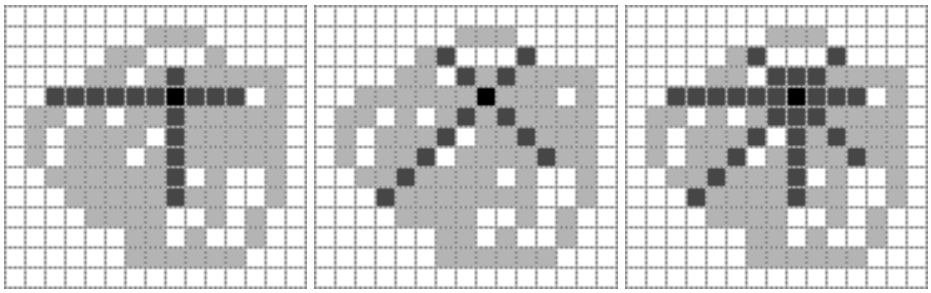
Portanto é possível calcular o número de possíveis configurações de XC β utilizando as funções braço de Θ. Uma vez que são omitidos os casos quando as funções braços $b_i \in \Theta$ se repetem e quando são mudadas de posição no vetor β, o método usado para calcular o número de tipos de XC é a combinação C_j^i (combinação de i, j a j) pois considera que nenhuma função se repete nem importa a ordem em que estão descritas:

$$C_j^i = \frac{i!}{j!(i-j)!},$$

em que i é o número de elementos que vão ser combinados e j é o número desses elementos que vão ser usados na combinação. No caso serão combinadas as 8 funções braços de Θ, logo $i = 8$. O número de elementos a ser combinado j é a dimensão do vetor β que é no mínimo 1 e no máximo 8, porque Θ só tem 8 elementos, logo, $1 \leqslant j \leqslant 8$.

O número de vetores β com uma dimensão é $C_1^8 = 8$, isto é, existem 8 configurações de XC que usam apenas uma função de Θ, por exemplo, $\beta = (b_L)$ ou $\beta = (b_S)$. O número de configurações de XC com duas dimensões é $C_2^8 = 28$, isto é, existem 28 configurações de XC $\beta \in \Theta^2$, por exemplo, $\beta = (b_L, b_O)$ ou $\beta = (b_S, b_O)$. O número de configurações de XC com três dimensões é $C_3^8 = 56$, isto é, existem 56 configurações de XC $\beta \in \Theta^3$, por exemplo, $\beta = (b_L, b_O, b_S)$ ou $\beta = (b_S, b_N, b_O)$. Do mesmo modo pode-se calcular o número de possíveis combinações para vetores β de 4 a 8 dimensões. Então o número total de configurações de vizinhança usando as fuções de Θ é $\sum_{j=1}^{8} C_j^8 = 255$.

Apesar de existirem 255 tipos de XC usando as funções de Θ foram escolhidas apenas 3 configurações para serem usadas neste trabalho. Os nomes desses três tipos de códigos são retirados das peças do xadrez torre, bispo e rainha, porque cada um desses códigos possui



(a) (b) (c)

Figura 3.4 Exemplo de vetores XC torre (a), bispo (b) e rainha (c). A célula preta da grade é o centro do XC, as células cinza escuro são os pontos representados por cada XC e as cinza claro são os demais pontos de pixels da imagem, as células brancas são os pixels do fundo. O vetor XC $\varphi^{torre} = (8,4,1,3,5,6)$ de (a) é do tipo torre. O vetor XC $\varphi^{bispo} = (8,4,2,3,5,2)$ de (b) é do tipo bispo. O vetor XC $\varphi^{rainha} = (8,4,1,2,3,3,5,5,6,2)$ de (c) é do tipo rainha.

vizinhança semelhante aos possíveis movimentos de cada uma dessas peças no jogo.

3.5.1 XC Tipo Torre

O vetor XC torre φ^{torre} codifica as vizinhanças acima, à direita, abaixo e à esquerda do centro do código utilizando as funções braços Norte (b_N), Leste (b_L), Sul (b_S) e Oeste (b_O). Como pode ser visto na Figura 3.4(a), os pontos codificados por esse tipo de XC assemelham-se aos pontos de possíveis jogadas de uma peça torre num jogo de xadrez. Essa configuração de código pode ser definida como um vetor no espaço $\mathbb{N}^6$:

$$\varphi^{torre} = (x,y,n,l,s,o) \in \mathbb{N}^6, \tag{3.20}$$

em que (x,y) é a posição do centro do código, n é o comprimento do braço para a função braço norte b_N, l é o comprimento do braço para a função braço leste b_L, s é o comprimento do braço para a função braço sul b_S e o é o comprimento do braço para a função braço oeste b_O. Essas funções braço compõem a configuração β^{torre} do φ^{torre}:

$$\beta^{torre} = (b_N, b_L, b_S, b_O). \tag{3.21}$$

O conjunto de pontos representado por um XC Torre φ^{torre} pode ser gerado através da

função vizinhaça $u^{torre} : \mathbb{N}^6 \to \wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing$ que faz uso da configuração β^{torre} na função u (eq. 3.17):

$$u^{torre}(\varphi) = u(\varphi^{torre}, \beta^{torre}) = b_N(x,y,n) \cup b_L(x,y,l) \cup b_S(x,y,s) \cup b_O(x,y,o). \tag{3.22}$$

3.5.2 XC Tipo Bispo

O vetor XC bispo φ^{bispo} codifica além as vizinhanças diagonais utilizando as funções nordeste (b_{NE}), sudeste (b_{SE}), sudoeste (b_{SO}) e noroeste (b_{SO}). Como pode ser visto na Figura 3.4(b), os pontos codificados por esse tipo de XC assemelham-se aos pontos de possíveis jogadas de uma peça bispo num jogo de xadrez. Essa configuração de código pode ser definida como um vetor no espaço $\mathbb{N}^6$:

$$\varphi^{bispo} = (x, y, n_e, s_e, s_o, n_o) \in \mathbb{N}^6, \tag{3.23}$$

em que (x,y) é a posição do centro do código, n_e é o comprimento do braço para a função braço nordeste b_{NE}, s_e é o comprimento do braço para a função braço sudoeste b_{SE}, s_o é o comprimento do braço para a função braço sudoeste b_{SO}, e n_o é o comprimento do braço para a função braço noroeste b_{NO}. Essas funções braço compõem a configuração β^{bispo} do φ^{bispo}:

$$\beta^{bispo} = (b_{NE}, b_{SE}, b_{SO}, b_{NO}). \tag{3.24}$$

O conjunto de pontos representado por um XC bispo φ^{bispo} pode ser gerado através da função vizinhaça $u^{bispo} : \mathbb{N}^6 \to \wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing$ que faz uso da configuração β^{bispo} na função u (eq. 3.17):

$$u^{bispo}(\varphi) = u(\varphi^{bispo}, \beta^{bispo}) = b_{NE}(x,y,n_e) \cup b_{SE}(x,y,s_e) \cup b_{SO}(x,y,s_o) \cup b_{NO}(x,y,n_o). \tag{3.25}$$

3.5.3 XC Tipo Rainha

O vetor XC rainha φ^{rainha} codifica além das vizinhanças diagonais, acima, à direita, abaixo e à esquerda do centro do código utilizando todas as funções de Θ. Como pode ser visto na Figura 3.4(c), os pontos codificados por esse tipo de XC assemelham-se aos pontos de possíveis

jogadas de uma peça rainha num jogo de xadrez. Essa configuração de código pode ser definida como um vetor no espaço $\mathbb{N}^{10}$:

$$\varphi^{rainha} = (x, y, n, n_e, l, s_e, s, s_o, o, n_o) \in \mathbb{N}^{10}, \tag{3.26}$$

em que (x, y) é a posição do centro do código, n é o comprimento do braço para a função braço norte b_N, n_e é o comprimento do braço para a função braço nordeste b_{NE}, l é o comprimento do braço para a função braço leste b_L, s_e é o comprimento do braço para a função braço sudoeste b_{SE}, s é o comprimento do braço para a função braço sul b_S, s_o é o comprimento do braço para a função braço sudoeste b_{SO}, o é o comprimento do braço para a função braço oeste b_O e n_o é o comprimento do braço para a função braço noroeste b_{NO}. Essas funções braço compõem a configuração β^{rainha} do φ^{rainha}:

$$\beta^{rainha} = (b_N, b_{NE}, b_L, b_{SE}, b_S, b_{SO}, b_O, b_{NO}). \tag{3.27}$$

O conjunto de pontos representado por um XC rainha φ^{rainha} pode ser gerado através da função vizinhaça $u^{rainha} : \mathbb{N}^{10} \to \wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing$ que faz uso da configuração β^{rainha} na função u (eq. 3.17):

$$u^{rainha}(\varphi) = u(\varphi^{rainha}, \beta^{rainha}) = b_N(x, y, n) \cup b_{NE}(x, y, n_e)$$

$$\cup b_L(x, y, l) \cup b_{SE}(x, y, s_e) \cup b_S(x, y, s) \cup b_{SO}(x, y, s_o) \cup b_O(x, y, o) \cup b_{NO}(x, y, n_o). \tag{3.28}$$

3.6 Reconstrução da Imagem a partir da Codificação de Vizinhança

Para reconstruir uma matriz de bits idêntica à imagem binária a partir da qual foi gerada a codificação de vizinhança são necessários:

1. as dimensões $w \times h$ da imagem;
2. a configuração $\beta = (b_1, b_2, \ldots, b_{d-2})$ de funções braço dos XCs;
3. e o cojunto $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_m\}$ de vetores XCs, em que m é o número de XCs que representam a imagem.

As dimensões $w \times h$ da imagem permitem construir uma matriz de pixels do tamanho da imagem original. A configuração β indica quais vizinhanças são utilizadas por cada XC. Cada $\varphi_i \in \Phi, 1 \leqslant i \leqslant m$ é um código de vizinhança e representa um conjunto de posições de pixels pretos na imagem, que podem ser gerados através da função de vizinhança $u(\varphi, \beta)$, equação 3.17.

A posição de todos os pixels pretos da imagem é recuperada do conjunto da codificação de vizinhança da imagem Φ. A função de vizinhança $v : (\wp(\mathbb{N}^d) \setminus \varnothing) \times \mathrm{B}^{d-2} \to \wp(\mathbb{N} \times \mathbb{N}) \setminus \varnothing$ gera todos os pontos pretos da imagem através da união dos conjuntos de pontos gerados por $u(\varphi_i, \beta)$ para cada XC:

$$v(\Phi, \beta) = \bigcup_{i=1}^{m} u(\varphi_i, \beta). \tag{3.29}$$

O número de pixels pretos gerados por $v(\Phi, \beta)$ pode ser descrito por $\left| v(\Phi, \beta) \right|$. Uma vez que v gera um conjunto e o operador $\left| . \right|$ representa a cardinalidade de um conjunto [Ros98], $\left| v(\Phi, \beta) \right|$ indica quantos elementos existem no conjunto $v(\Phi, \beta)$.

O processo para gerar a imagem a partir da codificação de vizinhança pode ser visto no algoritmo 3.1. O algoritmo recebe como entrada os elementos descritos acima e retorna como saída uma imagem binária representada através de uma matriz. O primeiro passo do algoritmo é alocar uma matriz P com as dimensões $w \times h$ da imagem final (linha 1). Depois cada posição p_{xy} da matriz P é preenchida com o valor *branco* (linha 4). Na linha 7 gerado um conjunto Π com todas as posições de pixels pretos a partir da função v (eq.3.29). Então as posições de Π são marcadas com *preto* na matriz P (linha 9). A última linha do algoritmo retorna a imagem binária representada através da matriz P.

3.7 Geração da Codificação de Vizinhança a partir da Imagem Binária

A codificação de vizinhança é uma representação de imagens binárias que transforma uma matriz de bits em um conjunto de códigos de vizinhaça (XC). Esta seção apresenta um algoritmo que faz o mapeamento entre os pixels pretos da imagem e os codigos de vizinhança que representam as posições desses pixels. Uma imagem exatamente igual pode ser reconstruída usando

Algoritmo 3.1: Reconstruindo a Imagem a partir da Codificação de Vizinhança

Entrada: Dimensões $w \times h$ da imagem final;

Entrada: Configuração do código de vizinhança β;

Entrada: Conjunto de XCs $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_m\}$;

Saída: Imagem binária P;

`// aloque uma matriz` P `de dimensões` $w \times h$

crie uma matriz $P = [p_{xy}]$ de dimensões $w \times h$;

`// pinte cada posição` p_{xy} `a matriz` P `de branco`

para $x = 0, 1, 2, \ldots, w-1$ **faça**

  para $y = 0, 1, 2, \ldots, h-1$ **faça**

    $p_{xy} \leftarrow$ branco;

  fim

fim

`// gere o conjunto` Π `de posições de pontos preto da imagem`

$\Pi = v(\Phi, \beta)$; `// equação 3.29`

`// pinte todos os pontos de` Π `na imagem`

para $(x, y) \in \Pi$ **faça**

  $p_{xy} \leftarrow$ preto;

fim

retorna *P;*

o algoritmo da Seção 3.6.

Para definir um código de vizinhança são necessários:

1. a posição (x,y) do centro;
2. a configuração do código $\beta = (b_1, b_2, \ldots, b_{d-2})$;
3. e o comprimento c_i de cada braço, $1 \leqslant i \leqslant d-2$, em que d é a dimensão do vetor XC.

A configuração β do código é um parâmetro que indica que tipo de código de vizinhança se quer usar. A posição (x,y) do centro do código é facilmente obtida da imagem, uma vez que cada pixel preto gera um centro. A partir de cada centro deve-se calcular o comprimento dos braços c_i em relação a cada função braço da configuração $b_i \in \beta$. O critério escolhido para definir o comprimentos dos braços é: escolher o comprimento do braço para cada função que maximize o número de pontos codificados pelo XC. Todos os pontos codificados por um XC formam um conjunto que pode ser obtido através da função vizinhança $u(\varphi,\beta)$ (equação 3.17), o número de elementos desse conjunto é a sua cardinalidade e é expresso da seguinte forma $\left|u(\varphi,\beta)\right|$. Portanto para calcular o comprimento dos braços de um XC φ_{xy}, dada a posição (x,y) do seu centro, deve-se maximizar $\left|u(\varphi,\beta)\right|$:

$$\varphi_{xy} = \underset{\varphi=(x,y,c_1,c_2,\ldots,c_{d-2})}{\arg\max} (\left|u(\varphi,\beta)\right|), u(\varphi,\beta) \subseteq \Pi \tag{3.30}$$

em que Π é conjunto de todas as posições dos pixels pretos de uma imagem binária P, (x,y) é a posição do centro fixado para φ_{xy}, as variáveis a serem maximizadas $(c_1, c_2, \ldots, c_{d-2})$ são os comprimentos de cada braço associado à configuração, $\beta = (b_1, b_2, \ldots, b_{d-2})$, d é a dimensão de φ_{xy} e $u(\varphi,\beta)$ está expresso na equanção 3.17.

O Algoritmo 3.2 descreve como gerar cada código de vizinhaça a partir de uma imagem binária representada como uma matriz. O algoritmo recebe como entrada uma imagem binária na forma de uma matriz $P = [p_{xy}]$ e uma configuração de XC β com pelo menos uma função braço. E dá como saída uma lista Φ de XCs dessa imagem conforme a configuração β definida, m XCs são gerados, um para cada pixel preto da imagem. Na linha 1 é criada uma lista Π onde são adicionados todas as posições de pixels pretos da imagem (linha 5). Para cada posição (x,y)

de pixel preto da imagem o algoritmo define um centro de um XC φ_{xy} e calcula o comprimento de cada braço c_i, maximizando a equação 3.30 (linha 10). Cada código gerado é adicionado ao conjunto de saída Φ (linha 11). Finalmente o algoritmo retorna a lista Φ com todos os XCs extraídos da imagem (linha 12).

3.8 Algoritmos para Redução do Conjunto de Códigos

Uma vez que cada vetor de vizinhança XC representa vários pixels e o Algoritmo 3.2 gera um XC para cada pixel preto da imagem, a codificação de vizinhança de uma imagem é retundante. Isto é, são gerados mais códigos de vizinhança para uma imagem do que o necessário para reconstruí-la sem perda.

A Figura 3.3 mostra um exemplo de codificação de vizinhança, ela possui 9 pixels pretos e 9 códigos XC são gerados para representar essa imagem, cada código está representado por uma letra de (a) a (i). Pode-se então representar a codificação de vizinhaça da Figura 3.3 (j) como o conjunto de código

$$\Phi = \{a,b,c,d,e,f,g,h,i\}.$$

Existem vários conjuntos reduzidos de código $\Phi'_i \subset \Phi$ capazes de reconstruir a imagem (j), por exemplo, $\Phi'_1 = \{c,f,g,i\}$ ou $\Phi'_2 = \{d,h,i\}$.

O objetivo desta seção é propor algoritmos para a redução do conjunto de códigos XC necessários para reconstruir sem perdas uma imagem binária. O primeiro desses algoritmo propostos é o REDAG, um algoritmo genético. RED1 e RED2 são os nomes dos outros algoritmos propostos, estes dois algoritmos são determinísticos, ao contrário do algoritmo genético que é estocástico.

3.8.1 Redução do Conjunto de Códigos Via Algoritmo Genético (REDAG)

Algoritmos Genéticos (AG) são algoritmos geralmente usados para problemas de otimização [ES03]. Os AGs foram propostos inicialmente em [Hol75]. Estes algoritmos são inspirados

Algoritmo 3.2: Gerando a Codificação de Vizinhança a partir de uma Imagem Binária

Entrada: Imagem binária $P = [p_{xy}]$ de dimensões $w \times h$;

Entrada: Configuração do código de vizinhança β;

Saída: Conjunto de XCs $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_m\}$, em que m é o número de pixels pretos da imagem;

crie uma lista Π;
para $x = 0, 1, 2, \ldots, w-1$ **faça**
  para $y = 0, 1, 2, \ldots, h-1$ **faça**
    `// identifica cada pixel preto da imagem`
    se $p_{xy} = preto$ **então**
      adicione (x, y) a Π;
    fim
  fim
fim
`//` Π `agora contém todas as posições de pixels pretos na imagem`
`// construa um código para cada pixel preto da imagem`
para $(x, y) \in \Pi$ **faça**
  `// calcula os comprimentos do braços que maximiza o número de pixels codificados pelo código com centro em` (x, y)
  $\varphi_{xy} = \underset{\varphi=(x,y,c_1,c_2,\ldots,c_{d-2})}{\arg\max} (\left|u(\varphi, \beta)\right|), u(\varphi, \beta) \subseteq \Pi$; `//` `equação` 3.30
  adicione φ_{xy} à lista de saída Φ;
fim
retorna Φ;

na teoria da evolução, a qual afirma que espécies menos aptas para sobrevivência em um determinado meio são extintas devido à seleção natural, indivíduos mais adaptados ao meio têm mais chance de gerar descendentes, passando seus genes para a próxima geração. Os genes dos indivíduos mais aptos se tornam dominantes na população com o passar das gerações. Outro processo que pode acontecer durante a evolução é a mutação genética, uma alteração nos genes de um indivíduo que pode fazê-lo tanto mais como menos apto à sobrevivência.

O funcionamento dos AGs se baseia em uma *população* que evolui ao longo das gerações. Uma geração é um ciclo de iteração de um AG. Uma população é um conjunto de indivíduos, onde cada *indivíduo* é uma possível solução do problema de otimização. A qualidade da solução é medida através de uma *função de aptidão*. A representação do indivíduo é chamada *cromossomo* e é formada por um conjunto de *genes*. A evolução da população é o crescimento da aptidão dos seus indivíduos. Embora a aptidão de um indivíduo não mude, a população evolui através da substituição de indivíduos menos aptos por outros mais aptos. A cada geração são produzidos novos indivíduos através da recombinação e mutação dos cromossomos de indivíduos da população atual. Estes novos indivíduos substituem os antigos e formam a próxima geração da população.

Os AGs são algoritmos estocásticos, muitas das tarefas no ciclo desses algoritmos baseiam-se em escolhas randômicas. Por este fato, um AG dificilmente gera o mesmo resultado em repetidas execuções.

Nesta seção é proposto o REDAG, um AG para a redução do número de códigos de vizinhança necessários para reconstruir uma imagem binária sem perdas. A representação usada para os indivíduos da população desse AG é um vetor de bits α, o *cromossomo* do indivíduo:

$$\alpha = (g_1, g_2, \ldots, g_m), g_i \in \{0,1\}, 1 \leqslant i \leqslant m, \tag{3.31}$$

em que g_i é um gene do cromossomo α e m é o número de códigos XC usados para representar uma imagem sem perdas. Se o algoritmo 3.2 é usado para extrair os códigos de vizinhança, m é o número de pixels pretos da imagem. Cada gene g_i está associado a uma função φ_i (eq. 3.15) de um vetor δ,

$$\delta = (\varphi_1, \varphi_2, \ldots, \varphi_m), 1 \leqslant i \leqslant m, \tag{3.32}$$

em que a dimensão de δ é a mesma dimensão m de α.

O conjunto de códigos representado por α em relação ao vetor δ é o indivíduo Φ_δ^α da população:

$$\Phi_\delta^\alpha = \{\varphi_i | g_i = 1, 1 \leqslant i \leqslant m\}, \tag{3.33}$$

ou seja, $\varphi_i \in \Phi_\delta^\alpha$ quando o valor de g_i em α é 1 e $\varphi_i \notin \Phi_\delta^\alpha$ quando valor de g_i em α é 0.

O objetivo do REDAG é selecionar $k < m$ dos códigos XC de δ, de modo que ainda seja possível reconstruir a imagem sem perdas usando apenas esses k códigos. O resultado do REDAG é tão melhor quanto menor for o valor de k, restrito à condição de reconstruir a imagem original sem perdas. Portando antes de se medir a qualidade da solução Φ_δ^α gerada é preciso verificar se esta solução é capaz de reconstruir a imagem original sem perdas. Seja $P = [p_{xy}]$, a imagem original, uma imagem binária representada na forma de uma matriz de dimensões $w \times h$, Φ o conjunto de todos os códigos de vizinhança gerados a partir de P e de uma configuração β pelo Algoritmo 3.2, seja δ um vetor com os códigos XC de Φ, α um cromossomo do REDAG, Φ_δ^α o conjunto dos códigos representado α em relação a δ e $P' = [p'_{xy}]$ uma imagem binária construída pelo Algoritmo 3.1 a partir da codificação Φ_δ^α, da configuração β e das dimensões $w \times h$ de P, diz-se que Φ_δ^α reconstrói P sem perdas quando a distância de Hamming [WM97, dC07] entre P e P' é zero. A distâcia de Hamming d entre duas matrizes binárias P e P' pode é calculada por:

$$d(P, P') = \sum \left| p_{xy} - p'_{xy} \right|. \tag{3.34}$$

A partir disto é possível definir a função de aptidão f de um indivíduo Φ_δ^α no REDAG:

$$f(\Phi_\delta^\alpha) = \begin{cases} m - \sum_{i=1}^{m} g_i, & \text{se } d(P, P') = 0 \\ -1, & \text{se } d(P, P') \neq 0 \end{cases}, \tag{3.35}$$

onde Φ_δ^α é o conjunto de códigos gerado para o cromossono α dado um vetor de códigos XCs δ, $g_i \in \{0, 1\}$ é o valor do i-ésimo gene do cromossomo α, m é a dimensão de α e δ, $d(P, P')$ é a distância de Hamming entre a imagem original P e a imagem P' gerada a partir de Φ_δ^α. Quando o vetor α possui o valor 1 para todas as posições g_i dos seu genes, ou seja, $\alpha = (1, \ldots, 1)$, o valor da sua aptidão do indivíduo Φ_δ^α é zero, $f(\Phi_\delta^\alpha) = 0$, ele é capaz de reconstruir a imagem sem perdas mas não eliminou nenhum XC. Quando k genes de α são marcados como zero e

é possível reconstruir a imagem original sem perda a partir de Φ_δ^α, a aptidão do indivíduo é k, $f(\Phi_\delta^\alpha) = k$, que significa que k códigos foram eliminados sem perdas para a representação da imagem. Porém, quando k genes de α são marcados como zero e não é possível reconstruir a imagem original sem perda a partir de Φ_δ^α, o indivíduo não é uma solução adequada para o problema e sua aptidão $f(\Phi_\delta^\alpha) = -1$, ou seja, é uma solução pior do que $\alpha = (1,\ldots,1)$.

O operador de recombinação usado no REDAG é o *crossover de 1 ponto*. Este operador parte de dois cromossomos pais e gera dois cromossomos filhos a partir da troca de parte dos seus vetores a partir de j, uma posição do vetor escolhida randomicamente. Dados dois cromossomos pais α_1 e α_2

$$\alpha_1 = (g_1, g_2, \ldots, g_m),$$

$$\alpha_2 = (h_1, h_2, \ldots, h_m),$$

geram dois novos cromossomos filhos α_3 e α_4

$$\alpha_3 = (g_1, g_2, \ldots, g_j, h_{j+1}, \ldots, h_m),$$

$$\alpha_4 = (h_1, h_2, \ldots, h_j, g_{j+1}, \ldots, g_m),$$

em que m é o número de genes de cada cromossomo, j é um número randômico, $1 \leqslant j < m$, g_i é cada um dos genes do cromosso α_1 e h_i é cada um dos genes do cromossomo α_2, $1 \leqslant i \leqslant m$. Os primeiros j genes do filho α_3 são os j primeiros genes de α_1 e os $(m-j)$ últimos genes de α_3 são os últimos $(j-m)$ genes de α_2. Os primeiros j genes do filho α_4 são os j primeiros genes de α_2 e os $(m-j)$ últimos genes de α_4 são os últimos $(j-m)$ genes de α_1. Um exemplo do *crossover* de um ponto pode ser visto na Figura 3.8.1

A mutação no REDAG é aplicada logo após a recombinação em cada filho gerado, mudando o valor de alguns genes do cromossomo do filho (Figura 3.8.1). Para cada gene g_i do cromossomo α seu valor é mudado com uma probabilidade de mutação condicionada pelo valor do gene $p_m(\text{mutar} g_i | g_i)$:

$$p_m(\text{mutar } g_i | g_i = 1) = 0,70,$$

$$p_m(\text{mutar } g_i | g_i = 0) = 0,07.$$

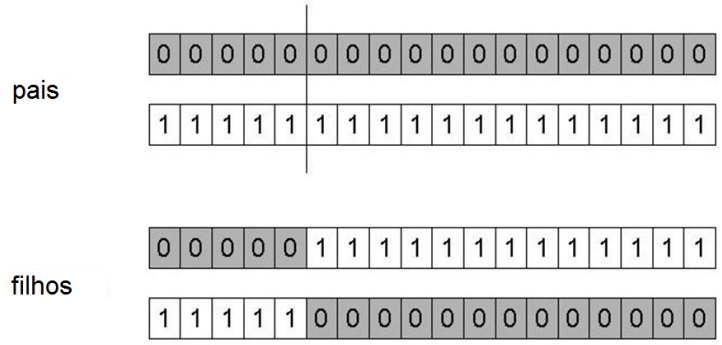


Figura 3.5 *Crossover* de 1 ponto. Um ponto j do cromosso é escolhido randomicamente, o vetor do cromossomo é dividido em dois segmentos: antes e depois do ponto j. Dois cromossomos pais que tiveram seus vetores divididos no ponto j trocam o segundo de seus segmentos dando origem a dois novos indivíduos chamados filhos.

Estes valores foram definidos empiricamente. A probabilidade de um gene de valor 1 mudar para 0 é de 70%, isto é, 70% de chance de excluir um código XC do indivíduo Φ_{δ}^{α}. E a probabilidade de mudar o valor de um gene de 0 para 1 é de 7%, isto é, 7% de chances de inserir um código XC no indivíduo Φ_{δ}^{α}. Durante a execução do algoritmo, um número pseudo-randômico r_i no intervalo [0,1] é gerado para cada posição do gene g_i, se r_i for menor que p_m, o gene g_i sofre mutação.

O REDAG está descrito pelo Algoritmo 3.3. A entrada é um conjunto Φ com m códigos de vizinhança e a saída é um conjunto reduzido desses códigos $\Phi' \subseteq \Phi$. O número de possíveis cromossomos para um determinado número m de códigos XC é 2^m. Para poder explorar mais o espaço de busca o tamanho da população no REDAG é proporcional a m. O número de cromossomos da população (linha 1) é $m/2$, 10000 é o tamanho máximo da população devido a restrições de recursos computacionais.

A população é inicializada (linha 2) com cada cromossomos contendo o valor 1 em todos os seus genes. O REDAG tem uma variável chamada melhor indivíduo (linha 3) que guarda o cromossomo de melhor aptidão ao longo de todas as gerações do algoritmo. A variável número de gerações sem evolução (linha 4) conta quantas gerações ocorreram desde a geração em que foi identidicado o melhor indivíduo. A linha 5 é a condição de parada do laço de gerações do REDAG, o processo é interrompido após 30 gerações sem evolução.

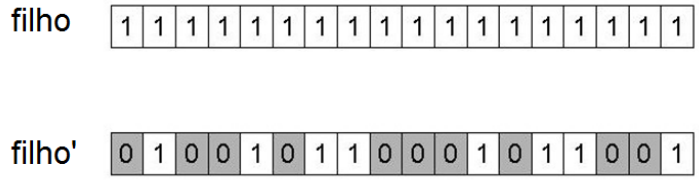


Figura 3.6 Exemplo de mutação. Cada gene do vetor cromossomo de um filho tem seu mudado com uma probabilidade de mutação p_m gerando um filho' diferente do inicial.

A linha 6 é o laço que repete o trecho que gera filhos, esse trecho é repetido $m/4$ vezes a cada geração, gerando 2 filhos em cada iteração, com um total de $m/2$ filhos por geração. A seleção dos pai é realizada por torneio (linhas 7 e 8). Este torneio seleciona randomicamente $m/100$ cromossomos da população e escolhe aquele com maior aptidão para ser o pai. Quando $(m/100) < 2$ pelo menos 2 cromossomos por torneio são selecionados de forma randômica.

Na linha 9, é realizada a recombinação através de *crossover* de 1 ponto entre o pai1 e o pai2, gerando dois filhos, filho1 e filho2. Em seguida (linhas 10 e 11) a mutação é aplicada a cada gene de cada filho da forma como foi descrita anteriormente. A linha 12 troca os nomes das variáveis para que o filho1 seja aquele que tem maior aptidão entre o filho1 e o filho2. Então, o filho1 tenta substituir o indivíduo de menor aptidão na população desde que a aptidão do filho1 seja maior do que a menor aptidão na população (linha 13). O mesmo se dá para o filho2 (linha 14).

Depois de gerados os filhos e substituídos na população, cada elemento da nova geração é verificado (linha 16). Se qualquer deles tiver aptidão maior do que a aptidão do melhor indivíduo (linha 17), esse cromossomo passa a ser o novo melhor indivíduo (linha 18) e o número de gerações sem evolução volta a ser zero (linha 19), pois a população evoluiu ao gerar um indivíduo de aptidão maior do que a do melhor indivíduo atual. Se não houve evolução o número de gerações sem evolução é incrementado em 1 (linha 21). O algoritmo termina retornando o conjunto de códigos XC representado pelo melhor indivíduo ao longo de todas as gerações (linha 24).

Algoritmo 3.3: REDAG: Redução do Conjunto de Códigos XC via Algoritmo Genético.

Entrada: Lista de codigos de vizinhança $\Phi = \{\varphi_i \,|\, i = 1, \ldots, m\}$;

Saída: Lista reduzida de codigos de vizinhança $\Phi' = \{\varphi_i \,|\, i = 1, \ldots, m'; m' \leqslant m\}$;

```
1  aloque uma lista população = (α_1, α_2, α_{m/2}) com min(10000, m/2) cromossomos;
2  para cada α_i da população faça α_i = (1, ..., 1);
3  aloque um cromossono chamado melhor indivíduo;
4  número de gerações sem evolução ← 0;
5  enquanto número de gerações sem evolução < 30 faça
6      para j = 1, ..., (m/4) faça
7          pai1 ← o cromossomo de maior aptidão entre max(2, m/100) escolhidos
           randomicamente da população;
8          pai2 ← o cromossomo de maior aptidão entre max(2, m/100) escolhidos
           randomicamente da população;
9          recombine pai1 e pai2 gerando os cromossomos filho1 e filho2;
10         aplique mutação ao filho1;
11         aplique mutação ao filho2;
12         se aptidão do filho2 > aptidão filho1 então temp ← filho1; filho1 ← filho2; filho1
           ← temp;
13         se menor aptidão da população < aptidão do filho1 então substitua o indivíduo
           de menor aptidão por filho1;
14         se menor aptidão da população < aptidão do filho2 então substitua o indivíduo
           de menor aptidão por filho2;
15     fim
16     para cada α_i da população faça
17         se aptidão de α_i > aptidão melhor indivíduo então
18             melhor indivíduo ← α_i;
19             número de gerações sem evolução ← 0;
20         fim
21         senão número de gerações sem evolução++;
22     fim
23 fim
24 retorna Φ' = Φ_δ^{melhor indivíduo} ;
```

3.8.2 Primeiro Algoritmo Determinístico para Redução do Conjunto de Códigos (RED1)

O RED1 (Primeiro Algoritmo Determinístico para Redução do Conjunto de Códigos), diferentemente do REDAG, gera sempre o mesmo conjunto reduzido de códigos para uma dada codificação de vizinhança, porque é um algoritmo determinístico.

Este método se baseia no conceito de **poder de codificação** ou poder de representação de um código XC. O poder de codificação de um XC é o número de pixels que o XC representa. Portanto, quanto maior o poder de codificação, maior a área que ele representa na imagem. O RED1 tenta excluir cada código XC de uma codificação, dando prioridade àqueles de menor poder de representação, um código só é mantido quando sua eliminação impossibilita a reconstrução completa da imagem.

O RED1 está descrito no algoritmo 3.4. Este método recebe como entrada um conjunto Φ de XCs, as dimensões $w \times h$ da imagem binária representada por esse conjunto de códigos e uma configuração β de XC. O resultado desse processo é um conjunto $\Phi' \subseteq \Phi$, geralmente [3] menor Φ e suficiente para representar a mesma imagem.

Este algoritmo varre o conjunto de códigos Φ em ordem crescente do número de pontos gerado por cada código. Ao processar cada código nesta ordem o algoritmo verifica se o código pode ou não ser descartado, se o código é identificado como essencial para a reconstrução sem perdas da imagem ele é adicionado ao conjunto de códigos final do algoritmo. A heurística utilizada para criar esse algoritmo é eliminar os códigos que representam menos pontos na imagem e manter aqueles que representam mais pontos.

A linha 1 do algoritmo 3.4 cria uma matriz $T = [t_{xy}]$ e inicia $T = 0$. Então percorre todas as posições de Φ (linha 2), gerando para cada posição φ_i todos os pontos (x, y) (linha 3). Para cada (x, y) o valor t_{xy} é incrementado (linha 4). Ao final desse processo a matriz T indicará quantas vezes cada posição (x, y) é representada pelos código de Φ.

Na linha 7, cria uma estrutura Φ' para guardar os códigos que serão selecionados de Φ e formarão a saída do algoritmo. Na linha 8, é criado Φ^{ord} que é uma lista ordenada em

[3] Se cada código XC de uma codificação de vizinhança representa apenas um pixel, nenhum código pode ser eliminado, portanto o conjunto reduzido de códigos $\Phi' \equiv \Phi$.

relação ao número de pontos gerado por cada φ_i. Φ^{ord} contém os mesmo elementos do conjunto Φ. O número de pontos gerados por φ_i é $|u(\varphi_i, \beta)|$. Em Φ^{ord} o vetor φ_i^{ord} gera tantos ou menos pontos que φ_{i+1}^{ord} pelo fato de ser uma lista ordenada, isso também pode ser expresso por $|u(\varphi_i^{ord}, \beta)| \leqslant |u(\varphi_{i+1}^{ord}, \beta)|$.

Na linha 9, o algoritmo percorre os códigos de Φ^{ord} começando pelo código que representa menos pontos ao que representa mais pontos. Na linha 10, esses pontos são gerados através da função de vizinhaça. Para cada um dos pontos (x, y) gerados, o valor de t_{xy} é decrementado (linha 11). Se ao ser decrementado o valor de t_{xy} passar a ser zero (linha 12), isso indica que nenhum XC φ_j^{ord} com $j > i$ codifica o ponto (x, y), portanto o código da iteração atual φ_i^{ord} deve ser adicionado à lista de códigos de saída Φ'. Enquanto $t_{xy} > 0$ existe φ_j^{ord} com $j > i$ que codifica (x, y) e será identificado nos passo seguintes da iteração. Ao final, o algoritmo retorna a lista de códigos selecionados Φ'.

3.8.3 Segundo Algoritmo Determinístico para Redução do Conjunto de Códigos (RED2)

O RED2 (Segundo Algoritmo Determinístico para Redução do Conjunto de Códigos) semelhantemente ao RED1 também baseia-se na idéia de poder de codificação do XC. Onde o poder de codificação é o número de pixels representados pelo XC. Enquanto o RED1 tenta eliminar os códigos que representam menos pixels, o RED2 guarda somente aqueles que representam mais pixels, com uma peculiaridade crucial: o poder de codificação de cada XC no RED2 é atualizado toda vez que um XC é selecionado.

No RED1 o poder de codificação de cada XC é o mesmo em todas as iterações do algoritmo. No RED2, cada vez que um XC é selecionado, todos os pixels representados por este código são marcados como codificados, e todos os outros códigos que representam algum desses pixels têm seu poder de codificação decrementado. Desta forma, a cada iteração do RED2, o poder de codificação de um XC é equivalente ao número de pixels representados pelo código menos o número de pixels já representados pelos códigos selecionados até aquela iteração.

O Algoritmo 3.5 descreve o funcionamento do RED2. Este método recebe como entrada

Algoritmo 3.4: RED1

Entrada: Lista de codigos de vizinhança $\Phi = \{\varphi_i \,|\, i = 1, \ldots, m\}$;

Entrada: Dimensões $w \times h$ da imagem;

Entrada: Configuração do código de vizinhança β;

Saída: Lista reduzida de codigos de vizinhança $\Phi' = \{\varphi_i \,|\, i = 1, \ldots, m'; m' \leqslant m\}$;

```
1  crie uma matriz T = [t_xy] de dimensões w × h,
2  inicie T = 0;
3  para i = 1, ..., m faça
4  |  para (x,y) ∈ u(φ_i, β) faça
5  |  |  t_xy ← t_xy + 1;
6  |  fim
7  fim
8  crie Φ' para guardar os códigos selecionados;
9  faça Φ^ord = {φ_i^ord ∈ Φ | u(φ_i^ord, β)| ⩽ |u(φ_{i+1}^ord, β)|};
10 para i = 1, ..., m faça
11 |  para (x,y) ∈ u(φ_i^ord, β) faça
12 |  |  t_xy ← t_xy − 1;
13 |  |  se t_xy = 0 então
14 |  |  |  adicione φ_i^ord a Φ';
15 |  |  fim
16 |  fim
17 fim
18 retorna Φ';
```

Algoritmo 3.5: RED2

Entrada: Lista de codigos de vizinhança $\Phi = \{\varphi_i \,|\, i = 1, \ldots, m\}$;

Entrada: Dimensões $w \times h$ da imagem;

Entrada: Configuração do código de vizinhança β;

Saída: Lista reduzida de codigos de vizinhança $\Phi' = \{\varphi_i \,|\, i = 1, \ldots, m'; m' \leqslant m\}$;

```
// PARTE 1:  INICIALIZAÇÃO DAS VARIÁVEIS
```

crie Φ'; inicie Φ' como uma lista vazia;

crie r; inicie $r = |v(\Phi, \beta)|$;

crie uma matriz $A = [a_{xy}]$ de dimensões imensões $w \times h$; inicie $A = 0$;

crie uma lista $D = \{d_i \,|\, i = 1, \ldots, m\}$; inicie $d_i = |u(\varphi_i, \beta)|$, $\forall \varphi_i \in \Phi, d_i \in D, 1 \leqslant i \leqslant m$;

```
// PARTE 2:  SELEÇÃO DOS CÓDIGOS
```

enquanto $r > 0$ **faça**

- $\varphi_l = \varphi_i \mid d_i \geqslant d_j \forall \varphi_i, \varphi_j \in \Phi$;
- adicione φ_l a Φ';
- $r \leftarrow r - d_l$;
- **para todo** $(x, y) \in u(\varphi_l, \beta)$ **faça**
  - **se** $a_{xy} = 0$ **então**
    - $a_{xy} = 1$;
    - **para todo** $d_k | (x, y) \in u(\varphi_k, \beta)$ **faça**
      - $d_k \leftarrow d_k - 1$;
    - **fim**
  - **fim**
- **fim**

fim

retorna Φ';

um conjunto Φ de XCs, as dimensões $w \times h$ da imagem binária representada por esse conjunto de códigos e uma configuração β de XC. O resultado desse processo é um conjunto $\Phi' \subseteq \Phi$, geralmente [4] menor Φ e suficiente para representar a mesma imagem.

Na linha 1, é criada e inicializada a lista Φ' que irá conter os códigos selecionados pelo algoritmo. Na linha 2, é criada e inicializada a variável r que conta quantos pixels da imagem ainda restam para ser codificados. Esta variável é inicializada com o número de pixels pretos $|v(\Phi,\beta)|$ (eq. 3.29) da imagem, pois nenhum pixel da imagem foi codificado ainda.

A matriz $A = [a_{xy}]$ é criada e inicializada na linha 3. Esta matriz marca as posições (x,y) dos pixels codificados por Φ'. Quando $a_{xy} = 0$ indica que o pixel (x,y) ainda não foi codificado, quando $a_{xy} = 1$ indica que o pixel (x,y) já foi codificado. Como nenhum pixel foi codificado, A é inicializado com zero. A lista D (linha 4) guarda o poder de codificação d_i de cada código XC $\varphi_i \in \Phi$. Como nenhum código foi selecionado ainda, d_i é inicializado como o número de pontos $|u(\varphi_i,\beta)|$ (eq. 3.17) gerado pelo XC φ_i.

O laço iterativo do RED2 continua enquanto $r > 0$, ou seja, enquanto existirem pixels não codificados na imagem (linha 5). A cada iteração desse laço, é selecionado o código φ_l com o maior poder de codificação d_l na iteração atual, isto é, o código escolhido é aquele que representa o maior número de pixels (linha 6). Este código φ_l é adicionado à lista de resposta Φ' do algoritmo (linha 7). E o número de pixels codificados r é decrementado do poder de codificação d_l do código φ_l, $(r \leftarrow r - d_l)$, na linha 8.

Então, para cada ponto (x,y) representado pelo código φ_l selecionado na iteração (linha 9), desde que o ponto não esteja marcado como codificado, $a_{xy} = 0$, (linha 10), são atualizadas: a matriz A que marca os pixels codificados e o poder de codificação d_k de cada código φ_k afetado com a inclusão de φ_l em Φ'. Cada pixel (x,y) representado por φ_l é marcado como codificado, $a_{xy} = 1$, na linha 11. E o poder de codificação de cada código φ_k que também codifica (x,y) (linha 12) é decrementado em uma unidade (linha 13).

O efeito de fazer a atualização do poder de codificação d_k a partir de cada ponto (x,y) codificado por φ_l é que a lista D de poderes de codificação indica exatamente quantos pixels

[4] Se cada código XC de uma codificação de vizinhança representa apenas um pixel, nenhum código pode ser eliminado, portanto o conjunto reduzido de códigos $\Phi' \equiv \Phi$.

cada XC é capaz de representar na imagem a cada iteração. Um exemplo disso é que o valor d_l, do código selecionado φ_l, é o máximo valor $d_i \in D$ no começo da iteração em que ele é escolhido e $d_l = 0$ ao fim da mesma iteração, ou seja, todos os pixels que φ_l é capaz de representar já estão codificados na imagem, e seu poder de representação é nulo nas iterações seguintes. Essa é a principal diferença entre o RED1 e o RED2, o poder de codificação de cada código XC no RED1 é fixo para todas as iterações do algoritmo.

3.9 Experimentos

Esta seção trata sobre os experimentos realizados para avaliar os algoritmos REDAG, RED1 e RED2 para a redução do conjunto de códigos XC de uma codificação de vizinhança. Os algoritmos são avaliados apenas pela qualidade das soluções que geram, ou seja, pela redução que conseguem fazer. Uma solução é melhor que outra quando ela consegue fazer uma maior redução do conjunto de códigos. Não é interesse desta dissertação avaliar os algoritmos em termos de tempo de processamento ou consumo de memória. Embora essas análises de custo sejam relevantes, o foco, aqui, é verificar o funcionamento dos algoritmos propostos.

Cada algoritmo é aplicado para a codificação de vizinhança de 24 imagens. A codificação de vizinhança de cada imagem gera um código XC para cada pixel preto da imagem através do Algoritmo 3.2. O REDAG é avaliado apenas para a codificação de vizinhança utilizando a configuração XC do tipo Torre, enquanto o RED1 e o RED2 são avaliados também usando as configurações tipo Bispo e Rainha.

3.9.1 Base de Imagens para Teste

O conjunto de imagens usado para teste é composto de 24 imagens. Essas imagens podem ser visualizadas na Figura 3.7, seus nomes e suas dimensões estão listadas na Tabela 3.1. Dessas images duas são *halftone*: cat e ouster. Três são imagens de texto: courier12, oldeng16 e times12i. Duas são imagens cuja área preta está na borda da imagem: bell-2 e bat-12. Dez são

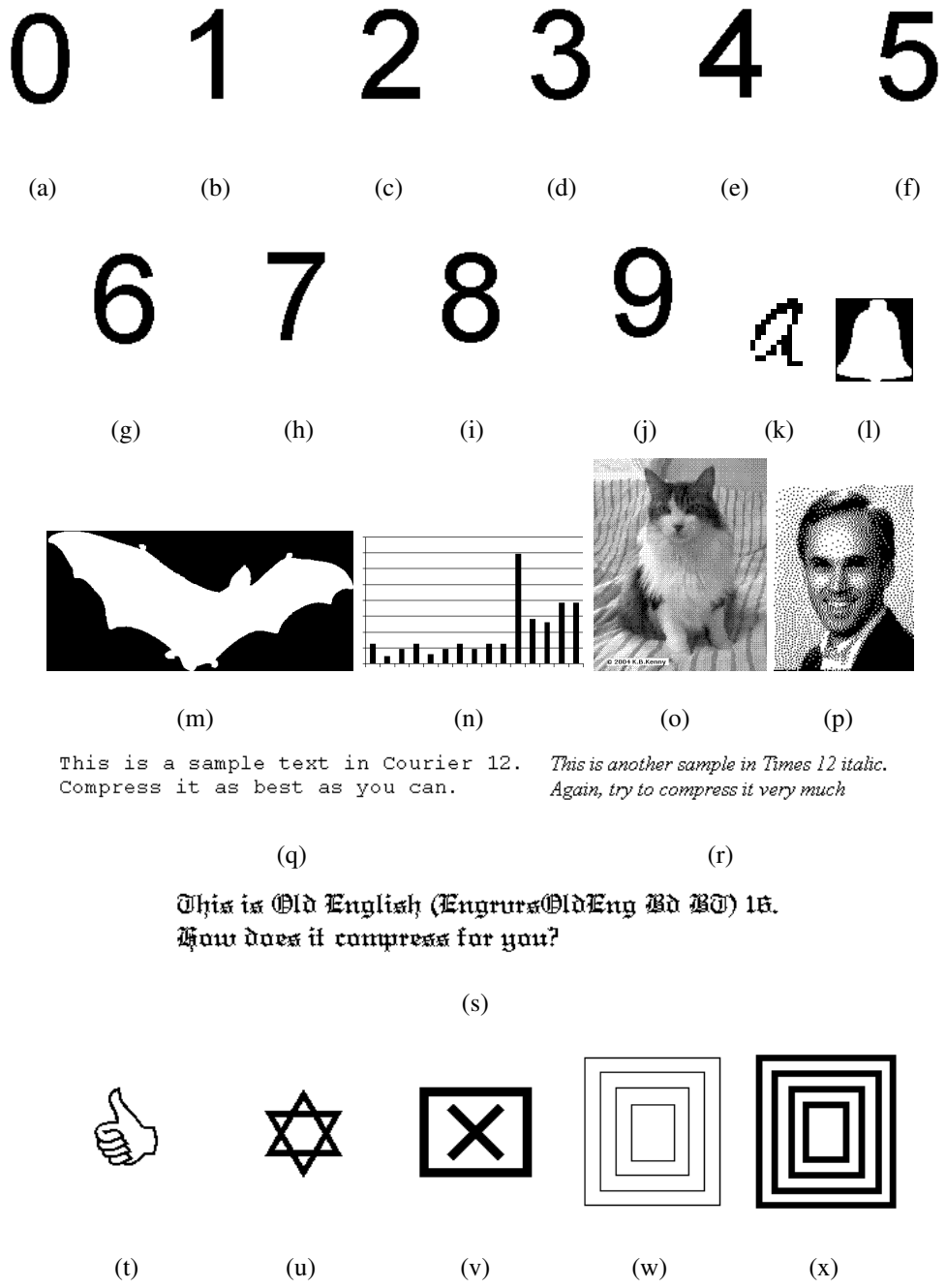


Figura 3.7 O nome de cada imagem do conjunto de testes e a escala que cada uma aparece nesta imgem. Quando a escala é 0,5× indica que as dimensões da imagem foram reduzidas para a metade. (a) *0*, 0,5×; (b) *1*, 0,5×; (c) *2*, 0,5×; (d) *3*, 0,5×; (e) *4*, 0,5×; (f) *5*, 0,5×; (g) *6*, 0,5×; (h) *7*, 0,5×; (i) *8*, 0,5×; (j) *9*, 0,5×; (k) *a*, 2×; (l) *bell-2*, 0,5×; (m) *bat-12*, 0,25×; (n) *grafico*, 0,25×; (o) *cat*, 0,175×; (p) *ouster*, 0,5×; (q) *courier12*, 0,5×; (r) *times12i*, 0,5×; (s) *oldeng16*, 0,5×; (t) *wingdings72-1*, 0,5×; (u) *wingdings72-2*, 0,5×; (v) *wingdings72-3*, 0,5×; (w) *retangulos-size1*, 0,5×; (x) *retangulos-size2*, 0,5×.

imagens de dígitos numéricos usando a fonte Arial no tamanho 72pt, são as imagem de 0 a 9. Uma é uma imagem de dimensões 17× do caracter a. E as outras seis imagens são desenhos digitais. Esse conjunto foi construído com uma variedade de imagens tal que fosse possível avaliar os algoritmos propostos sob várias condições distintas.

3.9.2 Testes com o REDAG

O REDAG é uma algoritmo estocástico, as atividades de mutação, recombinação e seleção dos pais, que são executadas nas iterações do algoritmo, são realizadas com base em números randômicos. Por esse fato, o resultado final do algoritmo pode variar em repetidas execuções. Portanto, o resultado de diversas execuções é utilizado para avaliar o REDAG. A variável utilizada nessa avaliação é o tamanho do conjunto de códigos selecionados, ou seja, número de códigos selecionados após redução.

Para cada imagem do conjunto de testes, o REDAG foi executado 100 vezes, com exceção da imagem bat-12 para qual o algoritmo foi executado apenas 29 vezes. Para o resultado das repetições foi calculado o mínimo e o máximo, a mediana, média e desvio padrão do número de códigos selecionados. Esses dados podem ser visualizados na Tabela 3.2. Por se tratar de uma algoritmo para redução de códigos, ao avaliar duas soluções, considera-se que a melhor é aquela que seleciona o menor número de códigos.

O REDAG não conseguiu produzir uma solução melhor do que o conjunto completo de códigos para as imagens cat, ouster, courrier12, times12i, oldeng e wingding72-1. Acredita-se que tal fato ocorreu devido à alta taxa de mutação para eliminar código. Pois todas essas imagens possuem traços desenhos com 1 ou 2 pixels de largura, isto causa uma baixa sobreposição dos pixels dos códigos XC que, juntamente com a alta taxa de eliminação de códigos, faz com que o REDAG gere apenas soluções que não são capazes de reconstruir a imagem sem perda, essas soluções não entram na população e são descartadas.

Para as demais imagens o conjunto de códigos é reduzido para aproximadamente 10% do original, considerando o mínimo valor alcançado, e 14%, considerando a média dos valores alcançados. A média e a mediana apresentam valores próximos para cada imagem, exceto

três (as imagens 3, a e grafico), o que indica que o REDAG tem um comportamento esperado regular na maioria dos casos. A razão desvio padrão pela média possui valores menores que 0,1 para todos os casos, exceto para as imagens 3, a e grafico, onde este valor é maior que 0,3 nos três casos. Essas imagens apresentaram grande variabilidade no tamanho de suas soluções, o que demonstra que o REDAG também pode ter um comportamento pouco regular.

A Tabela 3.3 mostra a média e mediana do número de gerações até encontrar a melhor solução de cada execução do algoritmo. Esta tabela foi medida exatamente para os mesmos experimentos a partir dos quais foi construída a Tabela 3.2. Para as imagens cat, ouster, courrier12, times12i, oldeng e wingding72-1 a geração em que encontrou a melhor solução é a primeira, pois nenhuma solução melhor que a inicial foi gerada. As demais imagens encontraram suas melhores soluções com menos de 30 gerações em média. Dez dessas imagens conseguiram achar a melhor solução com menos de 5 gerações em média.

3.9.3 Testes com o RED1

Os experimentos com o RED1 estão listados na Tabela 3.4. A coluna RED1T da tabela tem o resultado da redução para cada imagem codificada usando a configuração tipo Torre. A coluna RED1B da tabela tem o resultado da redução para cada imagem codificada usando a configuração tipo Bispo. A coluna RED1T da tabela tem o resultado da redução para cada imagem codificada usando a configuração tipo Rainha.

Os resultados de RED1T, se comparados com os melhores resultados do REDAG (coluna REDAG da mesma tabela), são melhores em quase todos os casos, com exceção das imagens 1, 4, a, bell-2 e retangulos-size2. A diferença entre esses dois resultados só é significante para duas imagens: 1 e retangulos-size2. Para essas mesmas duas imagens, mesmo os piores resultado do REDAG são melhores do que os do RED1. O que faz acreditar que o RED1 é melhor que o REDAG na maioria da vezes.

Comparando RED1T com RED1B, o segundo só supera o primeiro para os dois caso das imagens *halftone* cat e ouster. No caso da imagem cat, RED1B gera menos da metade de códigos do que RED1T. Nessas mesma duas imagens RED1R é quem gera a menor quantidade

de códigos. Embora seja de se esperar que RED1R gere no máximo tantos quanto RED1T ou RED1B, pois um XC Rainha representa os mesmo pontos que um XC Torre e que um XC Bispo para um mesmo centro (x,y) em uma mesma imagem P. Isto é, se um conjunto reduzido de XCs Torre ou Bispo é capaz de reconstruir uma imagem sem perdas, um conjunto de XCs Rainha com os mesmos centros dos XCs Torre ou dos XCs Bispo também é capaz de reconstruir a imagem. $u^{rainha}(\varphi) = u^{bispo}(\varphi) \cup u^{bispo}(\varphi)$, logo $u^{torre}(\varphi) \subseteq u^{rainha}(\varphi)$ e $u^{bispo}(\varphi) \subseteq u^{rainha}(\varphi)$. Apesar desse fato, o RED1R gera mais códigos do que o RED1T ou RED1B em 9 das 24 imagens de teste, são elas, 5, 6, 8, 9, bat-12, bell-2, courier12, retangulos-size1 e retangulos-size2. Isto é conseqüência do método de elininação do RED1 e da heurística que usa para estimar o poder de codificação de cada XC.

3.9.4 Testes com o RED2

Os experimentos com o RED2 estão listados na Tabela 3.4. A coluna RED2T da tabela tem o resultado da redução para cada imagem codificada usando a configuração tipo Torre. A coluna RED2B da tabela tem o resultado da redução para cada imagem codificada usando a configuração tipo Bispo. A coluna RED2T da tabela tem o resultado da redução para cada imagem codificada usando a configuração tipo Rainha.

Novamente, para as imagens *halftone*, a codificação Bispo apresentou maior redução do que a codificação Torre. RED2B gera menos códigos do que o RED2T para as imagens cat e ouster, como acontece com o RED1. Para as demais imagens, os conjuntos que usam a codificação Torre conseguem uma maior redução do que aquele que usam a codificação Bispo usando RED2. Como esperado, RED2R apresenta maior redução do que RED2T e REDEB, pois o código do tipo Rainha codifica mais pontos do que os códigos do tipo Bispo e Torre, como explanado anteriormente.

Considerando todos os casos testados, RED2T é melhor do que o RED1T e que o REDAG, o RED2B é melhor do que o RED1B e o RED2R é melhor que o RED1R. O algoritmo de redução RED2 demonstrou realizar maior redução que os demais algoritmo, portanto é a melhor solução conhecida até então para o problema.

3.9.5 Considerações Finais

Esta seção compara os algoritmos para redução de código REDAG, RED1 e RED2. Um aspecto observado é que o tipo de configuração de XC influencia diretamente o tamanho do conjunto de códigos reduzido, isto pode ser visto nos testes para os algoritmo RED1 e RED2 para os mesmo tipos de configuração.

Nos exemplos da base de dados fica mostrado também que o algoritmo RED2 consegue fazer a maior redução do conjunto de código do que os outros algoritmos propostos. Porém pode-se mostrar facilmente que RED2 não é ótimo através de um contra-exemplo: ao aplicar RED2 à codificação mostrada na Figura 3.3, o conjunto de códigos selecionados entre os 9 códigos extraídos da imagem é {c,e,g,i} enquanto o conjunto reduzido com menos códigos é {d,h,i}. Embora o REDAG tenha apresentado, nesses experimentos, os piores resultados dentre os três algoritmos, é provado em [Hol75] que este tipo de algoritmo é capaz de alcançar o resultado ótimo sujeito a restrições de tempo.

Tabela 3.1 Dimensões das imagens do conjunto de teste. *Largura* é o número de pixel na horizontal ou o número de colunas da imagem. *Altura* é o número de pixels na vertical ou o número de linhas da imagem.

Imagem	Largura	Altura
0	128	128
1	128	128
2	128	128
3	128	128
4	128	128
5	128	128
6	128	128
7	128	128
8	128	128
9	128	128
a	17	19
bat-12	474	216
bell-2	59	64
cat	380	469
courier12	374	46
grafico	349	211
oldeng16	476	55
ouster	108	144
retangulos-size1	129	136
retangulos-size2	129	136
times12i	278	46
wingdings72-1	129	136
wingdings72-2	129	136
wingdings72-3	129	136

Tabela 3.2 Redução do conjunto de códigos XC através do algoritmo REDAG. *Total* é o número de código antes da execução do REDAG. *Mínimo, Máximo, Mediana, Média* e *Desvio Padrão* são referentes ao tamanho do conjunto de códigos reduzido gerado pelo REDAG após 100 repetições para cada imagem.

Imagem	Total	Mínimo	Máximo	Mediana	Média	Desvio Padrão
0	1.349	113	161	138	138,0	10,17
1	756	46	66	56	56,2	3,42
2	1.313	140	231	196	193,8	15,66
3	1.222	166	385	339	279,0	88,28
4	1.328	90	123	112	111,8	5,39
5	1.376	137	224	192	187,0	22,15
6	1.524	193	266	231	231,0	13,64
7	910	69	88	80	79,7	3,67
8	1.536	195	469	230	235,7	38,92
9	1.498	182	254	224	223,9	12,24
a	45	15	45	22	27,8	11,78
bat-12	58.903	5.117	5.217	5.160	5.158,7	28,43
bell-2	1.558	118	142	133	132,8	4,30
grafico	9.558	1.271	2.891	1.346	1.491,0	446,88
cat	73.145	73.145	73.145	73.145	73.145,0	0,00
ouster	6.880	6.880	6.880	6.880	6.880,0	0,00
courier12	1.111	1.111	1.111	1.111	1.111,0	0,00
times12i	1.179	1.179	1.179	1.179	1.179,0	0,00
oldeng16	3.515	3.515	3.515	3.515	3.515,0	0,00
wingdings72-1	586	586	586	586	586,0	0,00
wingdings72-2	1.296	362	437	397	397,3	16,06
wingdings72-3	2.704	418	866	813	809,7	46,15
retangulos-size1	1.174	55	68	63	62,9	2,56
retangulos-size2	5.790	392	428	418	417,0	6,19

Tabela 3.3 *Mediana* e *Média* do número de gerações necessárias para encontrar o melhor resultado em cada repetição do REDAG. Estes resultados são referentes às mesmas execuções da Tabela 3.2.

Imagem	Mediana	Média
0	19	20,92
1	23	26,08
2	3	5,28
3	2	8,12
4	22	27,29
5	3	7,17
6	4,5	5,79
7	20	24,51
8	4	5,92
9	3	4,71
a	8	11,04
bat-12	28	33,58621
bell-2	26	30,47
grafico	7	10,02
cat	1	1
ouster	1	1
courier12	1	1
times12i	1	1
oldeng16	1	1
wingdings72-1	1	1
wingdings72-2	2	2
wingdings72-3	2	2,24
retangulos-size1	20	25,94
retangulos-size2	21	25,14

Tabela 3.4 Resultado da redução do conjunto de códigos XC. *Total* é o número de códigos antes da redução. O resultado do *REDAG* é o melhor entre 100 repetições, ver Tabela 3.2. *RED1T, RED1B, RED1R, RED2T, RED2B* e *RED2R* são os resultados da redução usando RED1 e RED2 com os códigos XC Torre (T), Bispo (B) e Rainha (R), respectivamente.

Imagem	Total	REDAG	RED1T	RED2T	RED1B	RED2B	RED1R	RED2R
0	1.349	113	139	39	165	88	128	38
1	756	46	78	16	87	62	77	12
2	1.313	140	85	50	163	75	74	33
3	1.222	166	111	59	154	79	106	47
4	1.328	90	94	33	131	83	87	23
5	1.376	137	103	48	195	98	111	41
6	1.524	193	142	57	207	96	145	50
7	910	69	68	33	120	69	42	24
8	1.536	195	134	60	194	87	145	54
9	1.498	182	140	54	201	93	144	48
a	45	15	19	13	20	16	18	9
bat-12	58.903	5.117	671	328	1.894	807	787	303
bell-2	1.558	118	123	38	245	150	131	35
grafico	9.558	1.271	745	150	3.656	3.544	790	147
cat	73.145	73.145	52.082	50.420	18.238	15.712	15.238	12.008
ouster	6.880	6.880	2.738	2.324	2.310	1.835	2.291	1.494
courier12	1.111	1.111	369	314	782	717	434	233
times12i	1.179	1.179	591	435	680	618	549	314
oldeng16	3.515	3.515	1.440	628	1.633	1.201	1.319	447
wingdings72-1	586	586	194	79	252	129	157	52
wingdings72-2	1.296	362	178	86	241	102	130	51
wingdings72-3	2.704	418	225	73	457	254	166	24
retangulos-size1	1.174	55	12	8	1.158	1.158	24	8
retangulos-size2	5.790	392	626	40	1.642	1.056	650	40

CAPÍTULO 4

Compressão de Imagens com Código de Vizinhança

Uma vez que uma imagem pode ser reconstruída a partir de sua codificação de vizinhança, é possível criar um formato de arquivo que use essa codificação. Diversos formatos de arquivo usando a codificação de vizinhança podem ser definidos. Para construir uma matriz de uma imagem representada através da codificação de vizinhança são necessários, segundo o Algoritmo 3.1: altura e largura da imagem; configuração β; e o conjunto de códigos XC que representam os pixels pretos da imagem. A configuração β pode ser usada para várias imagens distintas e portanto não precisa estar descrita no arquivo. Então um formato de arquivo usando a codificação de vizinhança deve conter a altura e a largura da imagem e todos os vetores de vizinhança necessários para representá-la.

Um formato de arquivo simples usando codificação de vizinhança pode usar 32 bits para cada número natural, este arquivo ocuparia $32 \times (2 + d \times m)$ bits na memória, em que d é o número de dimensões de cada vetor XC, m é o número de vetores XC que representam a imagem, os outros dois números que são representados são a altura e largura da imagem. Exemplo: a imagem bat-12 tem dimensões 474×216, a representação desta imagem como matriz pode ser feita usando apenas 1 bit para cada pixel. Portanto, essa imagem pode ser representada na memória com $474 \times 216 = 102.384$ bits. Utilizando o formato simples descrito acima, para os 58.903 códigos XC Torre gerados para a imagem bat-12 (Tabela 3.4), $32 \times (2 + 6 \times 58.903) = 11.309.440$ bits, o que é pior do que a representação da imagem usando 1 bpp (bit por pixel). De fato, essa representação usa mais de 110 bbp . Se, em vez de escrever no arquivo os 58.903 códigos gerados para bat-12, forem escritos apenas os 328 códigos gerados pelo algoritmo de redução RED2, o tamanho do arquivo final passa a ser $32 \times (2 + 6 \times 328) = 63.040$ bits. A redução do número de códigos através do RED2 gerou um compressão na imagem, pois

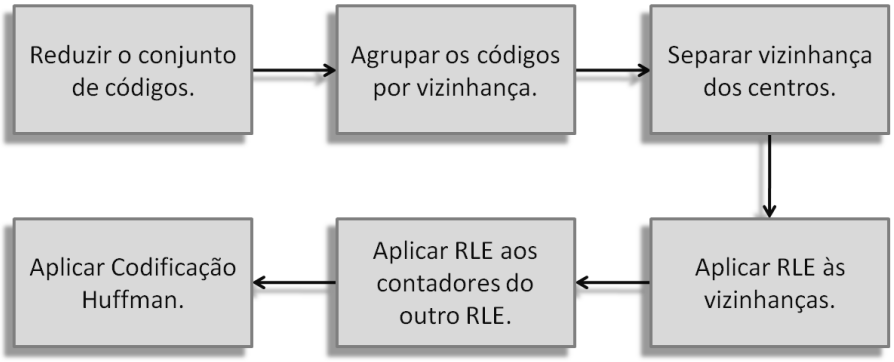


Figura 4.1 Diagrama de fluxo do método de compressão propostos. O processo é dividido em seis etapas: reduzir o conjunto de códigos, agrupar os códigos por vizinhança, separar a vizinhança dos centros, aplicar RLE às vizinhanças, aplicar RLE aos contadores do outro RLE e, por último, aplicar a codificação Huffman.

o arquivo final usa apenas 0,61 bpp.

Este capítulo propõe uma técnica para escrever a codificação de vizinhança num arquivo de forma compacta. Entende-se por compressão escrever um arquivo em um determinado formato que necessite de menos bits do que o formato original. A compressão de imagens usando codificação de vizinhança baseia-se na idéia de escrever num arquivo o conjunto reduzido de códigos XC. Para alcançar uma representação ainda mais compacta, são empregadas as técnicas de compressão: codificação Huffman e RLE (*Run Length Encoding*).

4.1 Método de Compressão

O método de compressão aqui proposto é dividido em seis partes, como mostrado no diagrama da Figura 4.1: reduzir o conjunto de códigos, agrupar os códigos por vizinhança, separar a vizinhança dos centros, aplicar RLE às vizinhanças, aplicar RLE aos contadores do outro RLE e, por último, aplicar a codificação Huffman. Cada uma dessas etapas é explicada a seguir.

Reduzir o conjunto de códigos de vizinhança. Essa primeira etapa do método de compressão tem uma importância crucial em eliminar os códigos redundantes de uma codificação de vizinhança. Três algoritmos para redução do conjunto de códigos estão explicados na Seção

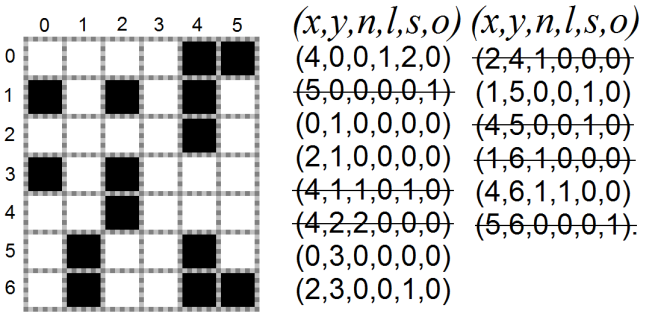


Figura 4.2 Uma imagem binária e os códigos XC tipo Torre que a representam. Os códigos riscados foram eliminados por algum algoritmo de redução de códigos. Com os códigos restantes ainda é possível reconstruir a imagem sem perdas

3.8. A Figura 4.2 mostra uma imagem binária e seus respectivos códigos XC tipo Torre. Os códigos que aparecem riscados na figura foram eliminados por algum algoritmo de redução do conjunto de códigos. Com os códigos restantes ainda é possível reconstruir a imagem sem perdas. Exemplo: o conjunto reduzidos de códigos para a imagem da Figura 4.2 é
(1,5,0,0,1,0) (0,1,0,0,0,0) (2,1,0,0,0,0) (4,0,0,1,2,0) (0,3,0,0,0,0) (2,3,0,0,1,0) (4,6,1,1,0,0).

Agrupar os códigos por vizinhança. O vetor XC é dividido em duas partes, a primeira parte é chamada centro e a segunda parte é chamada vizinhança. Esta etapa do método de compressão faz com que os vetores XC que tem a parte de vizinhança com os mesmo valores fiquem em seqüência na memória. Continuando o exemplo anterior, os códigos
(1,5,0,0,1,0) (0,1,0,0,0,0) (2,1,0,0,0,0) (4,0,0,1,2,0) (0,3,0,0,0,0) (2,3,0,0,1,0) (4,6,1,1,0,0),
passam a ficar nesta ordem, na memória,
(0,1,0,0,0,0) (2,1,0,0,0,0) (0,3,0,0,0,0) (2,3,0,0,1,0) (1,5,0,0,1,0) (4,0,0,1,2,0) (4,6,1,1,0,0).

Separar a vizinhança dos centros. Os vetores XC são repartidos em suas duas partes (centro e vizinhança). Então todos os centros são movidos para o começo e as vizinhanças para

o final da lista de números. Continuando o exemplo anterior, a cadeia
(0,1,0,0,0,0) (2,1,0,0,0,0) (0,3,0,0,0,0) (2,3,0,0,1,0) (1,5,0,0,1,0) (4,0,0,1,2,0) (4,6,1,1,0,0),
que pode ser interpretada desta outra maneira,
(0,1)(0,0,0,0) (2,1)(0,0,0,0) (0,3)(0,0,0,0) (2,3)(0,0,1,0) (1,5)(0,0,1,0) (4,0)(0,1,2,0) (4,6)(1,1,0,0),
é transformada nesta,
(0,1)(2,1)(0,3)(2,3)(1,5)(4,0)(4,6) (0,0,0,0)(0,0,0,0)(0,0,0,0)(0,0,1,0)(0,0,1,0)(0,1,2,0)(1,1,0,0),
que tem *m* posições de centros no começo e *m* vetores de braço no final. A ordem dos centros e das vizinhanças são equivalentes para que se possa recuperar os códigos XC.

Aplicar RLE às vizinhanças. RLE (*Run Length Encoding*) [MR96] é um algoritmo de compressão de dados que reduz o comprimento de cadeias de caracteres substituindo trechos dessa cadeia onde um símbolo se repete por pares símbolo e contador de repetições. Este contador indica o número de vezes que o símbolo se repete. Por exemplo, a cadeia
WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWW
WWWWWWWWWWWWBWWWWWWWWWWWWW
é transformada em
(W)12×,(B)1×,(W)12×,(B)3×,(W)24×,(B)1×,(W)14×.

Nessa etapa do método de compressão, o RLE trata cada vetor de braços como um símbolo. Cada série de vetores de braços iguais é substituída por apenas um vetor braço e um contador, que é um número indicando quantas vezes esse vetor se repete. Continuando o exemplo onde a cadeia dividida entre centro e código está descrita dessa maneira:
(0,1)(2,1)(0,3)(2,3)(1,5)(4,0)(4,6) (0,0,0,0)(0,0,0,0)(0,0,0,0)(0,0,1,0)(0,0,1,0)(0,1,2,0)(1,1,0,0).
O RLE aplicado às vizinhanças transforma a cadeia em:
(0,1)(2,1)(0,3)(2,3)(1,5)(4,0)(4,6) (0,0,0,0)3x (0,0,1,0)2x(0,1,2,0)1x (1,1,0,0)1x.

Aplicar RLE aos contadores do outro RLE. O número de contadores gerados pelo RLE quando aplicado aos vetores de braço pode se repetir bastante. Para reduzir a lista de contadores gerados, é aplicado RLE à lista de contadores gerada pelo RLE que foi aplicado aos vetores de

braços. Continuando o exemplo, a cadeia se encontra desta maneira:
(0,1)(2,1)(0,3)(2,3)(1,5)(4,0)(4,6) (0,0,0,0)3x (0,0,1,0)2x(0,1,2,0)1x (1,1,0,0)1x.
Esta cadeia é então reescrita para fazer com que a lista de contadores fique agrupada:
(0,1)(2,1)(0,3)(2,3)(1,5)(4,0)(4,6) (0,0,0,0)(0,0,1,0)(0,1,2,0)(1,1,0,0) (3x)(2x)(1x)(1x).
Então o RLE é aplicado à lista de contadores:
(0,1)(2,1)(0,3)(2,3)(1,5)(4,0)(4,6) (0,0,0,0)(0,0,1,0)(0,1,2,0)(1,1,0,0) (3x)1x,(2x)1x,(1x)2x.

Embora neste exemplo não seja possível notar a eficácia desta operação, em imagens com mais códigos a eficiência pode ser comprovada. Já no exemplo a seguir, é possível notar o funcionamento prático dessa etapa no método de compressão. Exemplo: a cadeia de contadores
(10×), (6×), (4×), (4×), (3×), (3×), (2×), (2×), (2×), (2×), (2×), (1×), (1×), (1×), (1×), (1×), (1×), (1×), (1×), (1×), (1×), (1×), (1×), (1×), (1×), (1×)
é reescrita como
(10×)1×, (6×)1×, (4×)2×, (3×)2×, (2×)6×, (1×)17×.

A principal característica dessa operação é reduzir o número de contadores iguais a 1. Estes contadores têm uma freqüência muito grande na codificação de uma imagem real, uma vez que ele indica um código que não se repete. Essa etapa foi adicionada ao método de compressão ao se verificar que a ocorrência de códigos que não se repetem é bastante comum.

Aplicar a codificação Huffman. Até esta etapa, o método de compressão apenas manipulou uma cadeia de números inteiros. Esta cadeia foi reduzida com a aplicação do RLE em duas etapas distintas. O passo final é escrever essa cadeia de número inteiros em um arquivo binário. A forma trivial de se fazer isso é transformar cada um desses números em uma cadeia de bits de tamanho fixo. Porém existem métodos como codificação Huffman e codificação Aritmética que são capazes de gerar uma cadeia de bits menor do que a cadeia gerada da forma trivial.

A codificação Huffman é usada para transformar a cadeia de inteiros que representa a codificação de vizinhança em uma cadeia de bits compacta. Esta codificação foi escolhida porque apresenta bons resultados de compressão além de ser de implementação bastante simples. A próxima seção explica como funciona e como foi implementada a codificação Huffman.

4.2 Codificação Huffman

Entende-se por dicionários a simples e rápida representação de um conjunto de símbolos através de códigos. Porém existe a preocupação em ocupar cada vez menos espaço na codificação desses símbolos, mesmo que isto represente mais custo computacional. Em substituição aos dicionários, foram formuladas codificações capazes de reescrever uma mesma informação usando menos símbolos do que a informação original [WMB99]. A codificação de Huffman é o método mais antigo e se manteve como um dos melhores por décadas. Com a primeira publicação na década de 1950 [Huf52] só foi superado posteriormente pela Codificação Aritmética [WNC87] em meados de 1970.

Suponha um alfabeto de 256 símbolos, um dicionário pode fazer a correspondência de cada símbolo a um código de 8 bits. Certamente a codificação seria muito rápida, porém o resultado não será o mais econômico no ponto de vista de memória. Fazendo o uso da freqüência de cada símbolo num texto é possível determinar códigos de forma que diminua esse valor, para este exemplo é possível usar apenas 4 bits para os 15 símbolos mais comuns e 12 bits para o restante.

Essa é a idéia base da Codificação de Huffman. Seu algoritmo é muito simples, resume-se a atribuir códigos menores para símbolos mais freqüentes e códigos maiores para os menos usados. Essa atribuição é feita dispondo da freqüência de cada símbolo e montando uma árvore de forma que os caminhos da raiz às folhas determinem o código para cada símbolo. É usada uma árvore binária onde cada símbolo é uma folha e os nós têm dois filhos cujas arestas recebem o valor 0 ou 1.

Uma execução do Algoritmo 4.1 pode ser vista na Figura 4.3. Os símbolos e freqüência usados neste exemplo estão descritos na Tabela 4.1. As iterações do algoritmo estão descritas abaixo:

Iteração 1: no começo do algoritmo cada objeto contém um símbolo e uma freqüência, como pode ser visto na Figura 4.3(a). Os dois primeiros símbolos selecionados são os símbolos 9 e 14, estes objetos apresentam as menores freqüências, 0,1 cada. Então é criado um novo objeto, que é um nó que contém os dois objetos selecionados. A freqüência desse nó é 0,2, a soma das

Algoritmo 4.1: Gerando a árvore da codificação Huffman.

Entrada: Um conjunto T com n objetos, onde cada objeto contém um símbolo e a freqüência desse símbolo.

Saída: Uma tabela que relaciona cada símbolo a um código.

1 **repita**

2 defina m_0 e m_1, os dois objetos de T com menor freqüência;

3 remova m_0 e m_1 de T;

4 adicione um novo objeto em T que tem referência r_0 para m_0 e r_1 para m_1 como, a freqüência desse novo objeto é a soma das freqüências de m_0 e m_1;

5 **até** *T conter apenas um objeto* ;

6 percorra a árvore da raiz até cada folha, em uma folha está um símbolo e o caminha de uma raiz até a folha é código do símbolo;

7 monte a tabela que relaciona cada símbolo a um código;

8 **retorna** *a tabela que relaciona cada símbolo a um código*

Tabela 4.1 Tabela de símbolos numéricos e suas respectivas freqüências em uma determinada cadeia. A coluna da direita mostra o código gerado para cada.

Símbolo	Freqüência	Código Huffman
0	0,35	00
1	0,25	01
3	0,20	10
9	0,10	110
14	0,10	111

freqüências dos dois objetos selecionados. Pelo ramo 0 desse nó, se chega ao símbolo 9 e pelo ramo 1 se chega ao símbolo 14. O resultado dessa iteração do algoritmo está explicitado na Figura 4.3(b).

Iteração 2: os nós selecionados anteriormente não podem ser mais escolhidos. As menores freqüências no conjunto de objetos a serem selecionados são 0,2 para o nó criado na iteração anterior e 0,2 para a folha que representa o símbolo 3. Esses dois objetos passam a ser filhos de um nó de freqüência 0,4. Pelo ramo 0 desse nó se chega à folha que contém o símbolo 3, e pelo ramo 1 do mesmo nó se chega ao nó criado na iteração anterior. O resultado dessa iteração pode ser visto na Figura 4.3(c).

Iteração 3: as freqüências dos objetos que ainda podem ser escolhidos são 0,35, 0,25 e 0,4. Os nós que representam os símbolos 0 e 1 são os de menor freqüência. Um novo nó é criado, pelo ramo 0 deste nó se chega ao símbolo 0, pelo ramo 1 se chega ao símbolo 1. O resultado dessa iteração pode ser visto na Figura 4.3(d).

Iteração 4: só restam dois objetos a serem escolhidos, eles então são agrupados em um nó raiz. O resultado dessa iteração pode ser visto na Figura 4.3(e).

A etapa seguinte do algoritmo é varrer a árvore e montar a tabela que associa cada símbolo a uma seqüência de bits. Saindo da raiz pelos ramos 0 e seguindo o ramo 0 do nó seguinte chega-se ao símbolo zero. Saindo da raiz pelos ramos 0 e seguindo o ramo 1 do nó seguinte chega-se ao símbolo 1. Saindo da raiz pelos ramos 1 e seguindo o ramo 0 do nó seguinte chega-se ao símbolo 3. Saindo da raiz pelos ramos 1, seguindo o ramo 1 do nó seguinte e o ramo 0 do outro nó chega-se ao símbolo 9. Saindo da raiz pelos ramos 1, seguindo o ramo 1 do nó seguinte e o ramo 1 do outro nó chega-se ao símbolo 14. A Tabela 4.1 mostra o código Huffman gerado para cada símbolo. Os símbolos mais freqüentes usam dois bits e os menos freqüentes usam 3 bits na sua representação.

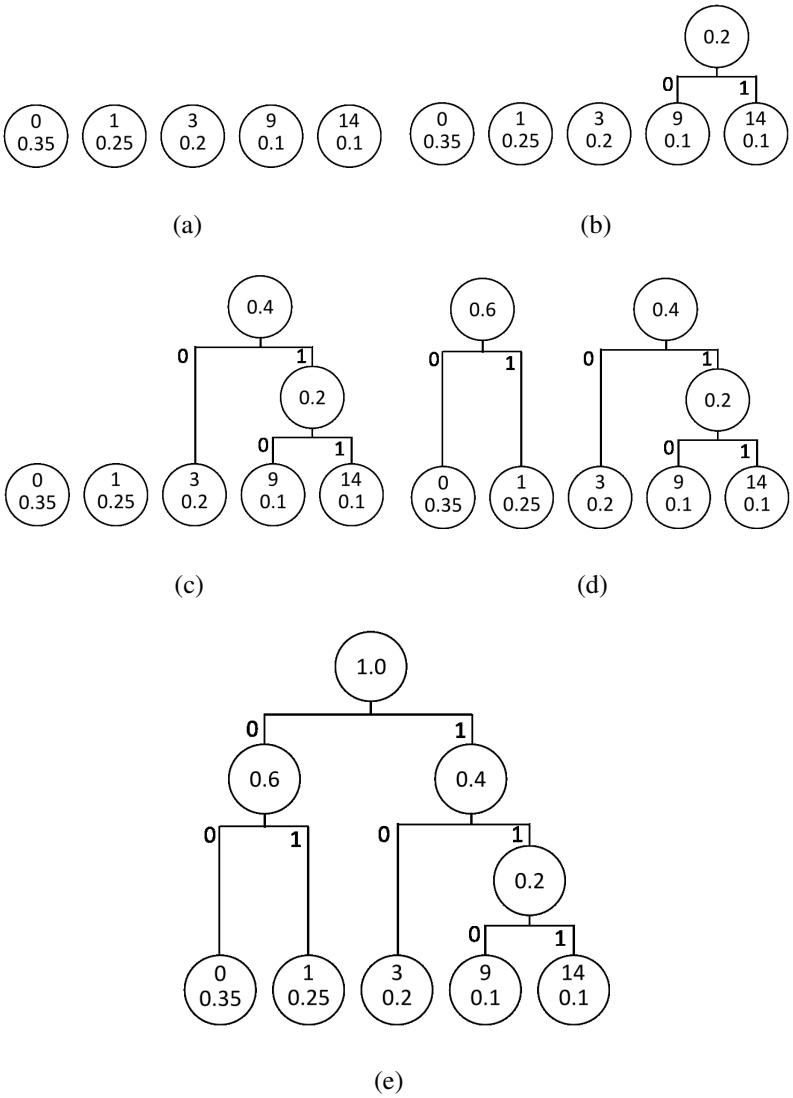


Figura 4.3 Exemplo da construção da árvore da codificação Huffman. Em (a) cada um dos símbolos está associado a sua freqüência. Em (b) os símbolos de menor freqüência são agrupados em um nó, a freqüência do nó e a soma das freqüências dos objetos agrupados. Em (c) o novo nó agrupa os objetos de menor freqüência que ainda não estão agrupados. O processo continua em (d) e a árvore está completamente construída em (e).

4.3 Formato do Arquivo

O método de compressão proposto neste capítulo é baseado no conceito de codificação de vizinhança. Uma cadeia de inteiros é gerada, esta cadeia contém todos os vetores XC. O comprimento dessa cadeia é reduzido através da aplicação do RLE. E essa cadeia é transformada em uma cadeia binária através da codificação Huffman. Porém resta especificar o formato do arquivo, que descreve: como é escrita a representação binária as informações que permitem fazer a decodificação Huffman, que tipo de configuração XC está sendo usada e as dimensões da imagem.

Os dois primeiros bits do arquivo indicam que tipo de configuração XC é empregada na codificação de vizinhança. Quando o valor desses dois bits é 10, a configuração utilizada é do tipo Torre. Quando é 01, a configuração é do tipo Bispo. E quando é 11 é do tipo Rainha. Os próximos 32 bits representam a largura e a altura da imagem, cada uma dessas medidas está escrita no arquivo como número binário de 16 bits.

Em seguida está escrito no arquivo uma representação binária da árvore binária gerada no Algoritmo 4.1 e uma lista dos símbolos representados por essa árvore. A representação binária da árvore é construída da seguinte forma: a árvore é percorrida em largura, isto é, os nós do próximo nível só são visitados depois que todos os nós do nível anterior a ele já foram visitados, como pode ser visto na Figura 4.4; em seguida cada elemento visitado na árvore é escrito no arquivo, um bit 1 é utilizado para indicar que o elemento visitado é um nó e um bit 0 é utilizado para indicar que o elemento visitado é uma folha. No exemplo da Figura 4.4, essa árvore é representada pela seqüência a=1 (nó), b=1 (nó), c=1 (nó), d=0 (folha), e=0 (folha), f=0 (folha), g=1 (nó), h=0 (folha), i=0 (folha), e escrita no arquivo como a seqüência de bits 111000100.

Os próximo 4 bits depois da árvore são a representação binária de um número s que indica quantos bits são usado para cada símbolo. Imediatamente depois está a lista de símbolos na ordem em que aparecem na cadeia que representa a árvore. No exemplo, essa lista de símbolos seria d=0, e=1, f=3, h=9, i=14. Cada um desses símbolos é um número representado por s bits. No exemplo $s = 4$, escrito no arquivo como $s = 0100$, e a representação binária dessa lista seria d=0000, e=0001, f=0011, h=1001, i=1110. A representação binária da árvore pode ser vista na

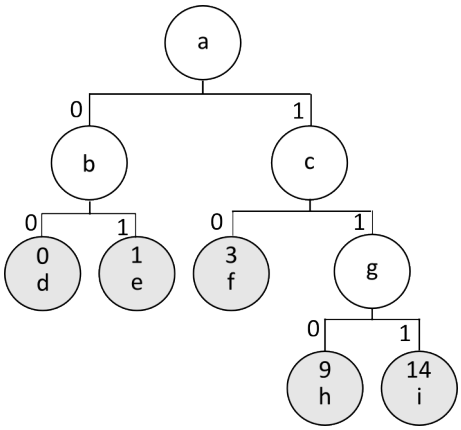


Figura 4.4 Ordem de leitura da árvore Huffman gerada pelo Algoritmo 4.1. A árvore é percorrida em largura, isto é, os nós do próximo nível só são visitados depois que todos os nós do nível anterior a ele já foram visitados. Esta ordem de leitura define como a árvore é escrita no arquivo.

árvore	$s = 4$	d=0	e=1	f=3	h=9	i=14
111000100	0100	0000	0001	0011	1001	1110 ...

Figura 4.5 Representação binária da árvore descrita na Figura 4.4. O primeiro trecho é uma representação binária da árvore que usa bits 1 para representar um nó e bits 0 para representar uma folha. s tem tamanho fixo de 4 bits e representa o número de bits usado para representar o símbolo de cada folha da árvore. Cada símbolo d, e, f, h, i, é presentado por s bits.

Figura 4.5.

No restante do arquivo é escrita a cadeia de inteiros que representa o conjunto de códigos de vizinhança. Cada símbolo é escrito como o código Huffman gerado para ele. Seguindo o exemplo anterior a cadeia de códigos

0,0,0,1,1,0,0,1,3,9,3,0,1,3,1,14

é escrita como

00 00 00 01 01 00 00 01 10 110 10 00 01 10 01 111,

utilizando para a conversão a Tabela 4.1.

4.4 Experimentos

A base de dados utilizada nos experimentos apresentados aqui é a mesma utilizada na Seção 3.9. Os métodos de redução testados foram o REDAG com a configuração Torre, o RED1 com a configuração Torre (RED1T), o RED1 com a configuração Bispo (RED1B), o RED1 com a configuração Rainha (RED1R), o RED2 com a configuração Torre (RED2T), o RED2 com a configuração Bispo (RED2B) e o RED2 com a configuração Rainha (RED2R).

Os resultados dos experimentos estão expressos em bpp (bits por pixel). Assume-se que a representação padrão de uma imagem binária utiliza 1 bpp para representar a imagem na forma de uma matriz. Portanto existe compressão numa imagem binária quando se consegue representá-la com menos de 1 bpp. E existe expansão quando a representação utiliza mais de 1 bpp. Houve compressão em 21 das 24 imagens para todos os algoritmos de redução. Nas outras 3 imagens – a, cat e ouster – nenhum algoritmo foi capaz de comprimi-las, o arquivo gerado após a codificação utiliza mais de 1 bpp para representar essas imagens. Com base nos dados da Tabela 4.2, pode-se perceber que o algoritmo RED2T obteve o melhor resultado de compressão para 10 das 24 imagens e o segundo melhor resultado em 11 dessas imagens. Por sua vez, o algoritmo RED2R obteve o melhor resultado em 15 imagens – convém ressaltar que na imagem retângulos-size1 houve empate do melhor resultado entre os algoritmos RED2T e RED2R – e obteve o segundo melhor resultado em 9 das imagens. Por esses dois algoritmos demonstrarem os melhores resultados para as imagens foi realizado um teste t emparelhado para verificar a igualdade entre as médias de bpp alcançadas pelo RED2T e RED2R, μ_{RED2T} e μ_{RED2R}. Para tanto, foram utilizados resultados da Tabela 4.2. As hipóteses do teste são:
H_0: $\mu_{RED2T} - \mu_{RED2R} = 0$, ou seja, as médias dos resultados de RED2T e RED2R são iguais
H_1: $\mu_{RED2T} - \mu_{RED2R} \neq 0$ RED2R, ou seja, as médias dos resultados de RED2T e RED2R são diferentes.
O p-valor do teste foi aproximadamente 0,19, o que indica que não se rejeita a hipótese nula a um nível de significância 10%. Portanto não se pode afirmar que RED2T e RED2R apresentam resultados diferentes.

Na Tabela 4.3, o método de compressão proposto, utilizando o algoritmo de redução RED2

com as configurações Torre (RED2T) e Rainha (RED2R), é comparado com os resultados da compressão dos formatos CCITT Group 3 e Group 4, PNG, GIF e JBIG. Os resultados estão expressos em bpp (bits por pixel). O método de compressão do JBIG obteve o melhor resultado em 21 das 24 imagens analisadas, sendo superado nos três casos restantes pelo RED2R.

Colocando em evidência os resultados do método proposto constata-se que o algoritmo RED2T obteve o segundo melhor resultado em 7 casos e empatou com o algoritmo RED2R para a imagem retangulos-size1. Por sua vez, o algoritmo RED2R, como mencionado, obteve o melhor resultado nas imagem 1, a e retangulos-size1. E obteve o segundo melhore resultado em 6 outros casos. Logo, tem-se que, juntos, RED2T e RED2R obtiveram o melhor ou segundo melhor desempenho na compressão em 15 casos, valor este superior ao alcançado pelos demais algoritmos com exceção do JBIG.

Todas as imagens nas quais o método proposto apresentou bons resultados têm em comum a característica de possuírem grandes áreas contínuas de pixels pretos, tais como imagens de desenhos e logotipos. A imagem bat-12 também possui grandes áreas contínuas de pixels pretos mas não apresentou resultado de compressão esperado, isto deve-se ao fato de a imagem apresentar dimensões muito grandes, demandando um conjunto grande de códigos XC. Apesar de o método proposto mostrar-se adequado para imagens de desenhos, apresentou-se não aconselhável para imagens *halftone* – cat e ouster – ou texto – courier12, times12i e oldeng16.

Tabela 4.2 Resultados do método de compressão proposto. Os resultados estão expressos em bpp (bits por pixel). Os métodos de redução testados foram o REDAG com a configuração Torre, o RED1 com a configuração Torre (RED1T), o RED1 com a configuração Bispo (RED1B), o RED1 com a configuração Rainha (RED1R), o RED2 com a configuração Torre (RED2T), o RED2 com a configuração Bispo (RED2B) e o RED2 com a coniguração Rainha (RED2R). Os resultados dessa tabela foram medidos no mesmo experimento que gerou os dados da Tabela 3.4. Os resultados para o REDAG são os melhores entre 100 execuções do algoritmo. O símbolo * indica o melhor resultado da compressão. O símbolo ** indica o segundo melhor resultado da compressão.

Imagens	REDAG	RED1T	RED2T	RED1B	RED2B	RED1R	RED2R
0	0,298	0,376	0,139*	0,392	0,240	0,423	0,168**
1	0,132	0,229	0,067**	0,208	0,158	0,259	0,059*
2	0,354	0,242	0,174**	0,380	0,216	0,269	0,149*
3	0,407	0,302	0,200**	0,383	0,233	0,361	0,199*
4	0,245	0,268	0,128**	0,296	0,235	0,303	0,118*
5	0,351	0,289	0,170*	0,452	0,278	0,377	0,181**
6	0,473	0,376	0,200*	0,498	0,267	0,480	0,211**
7	0,194	0,209	0,128**	0,279	0,198	0,166	0,110*
8	0,475	0,353	0,204*	0,471	0,253	0,479	0,224**
9	0,447	0,370	0,190*	0,482	0,267	0,471	0,203**
a	1,337**	1,882	1,412	1,783	1,536	1,981	1,214*
bat-12	2,486	0,332	0,225*	0,794	0,425	0,511	0,265**
bell-2	1,153	1,333	0,528*	2,085	1,430	1,718	0,581**
grafico	0,777	0,464	0,128*	1,361	1,371	0,564	0,146**
cat	12,299	8,347	8,154	3,239	2,902**	3,199	2,565*
ouster	14,807	5,213	4,297	4,597	3,613**	5,423	3,178*
courier12	1,896	0,745	0,642**	1,317	1,221	0,932	0,539*
times12i	2,487	1,503	1,175**	1,637	1,495	1,546	0,950*
oldeng16	4,299	2,052	0,976**	2,084	1,572	2,138	0,813*
wingdings72-1	1,012	0,406**	0,192	0,470	0,262	0,400	0,157*
wingdings72-2	0,752	0,405**	0,236	0,493	0,237	0,395	0,182*
wingdings72-3	0,922	0,518**	0,204	0,769	0,514	0,480	0,102*
retangulos-size1	0,144	0,033**	0,026*	1,518	1,518	0,061	0,026*
retangulos-size2	0,899	1,492	0,143*	2,621	1,885	1,671	0,155**

Tabela 4.3 Comparação entre os métodos de compressão propostos usando a redução de códigos RED2 com as conigurações Torre (RED2T) e Rainha (RED2R) e os formatos CCITT Group 3 e Group 4, PNG, GIF e JBIG. Os resultados estão expressos em bpp (bits por pixel).

Imagens	RED2T	RED2R	Group 3	Group 4	PNG	GIF	JBIG
0	0,139**	0,168	0,625	0,214	0,258	0,208	0,089*
1	0,067	0,059*	0,580	0,198	0,215	0,168	0,066**
2	0,174	0,149**	0,587	0,211	0,271	0,177	0,085*
3	0,200	0,199	0,596	0,220	0,291	0,188**	0,098*
4	0,128	0,118**	0,596	0,205	0,244	0,188	0,073*
5	0,170**	0,181	0,593	0,214	0,263	0,184	0,091*
6	0,200**	0,211	0,619	0,222	0,297	0,208	0,102*
7	0,128	0,110**	0,578	0,199	0,225	0,160	0,069*
8	0,204**	0,224	0,615	0,222	0,271	0,204**	0,095*
9	0,190**	0,203	0,616	0,223	0,294	0,208	0,097*
a	1,412**	1,214*	10,006	8,669	6,960	1,783	1,486
bat-12	0,225	0,265	0,177	0,064**	0,098	0,112	0,032*
bell-2	0,528**	0,581	1,481	0,917	0,831	0,538	0,314*
grafico	0,128	0,146	0,354	0,087**	0,099	0,207	0,023*
cat	8,154	2,565	2,950	3,132	0,365**	0,473	0,283*
ouster	4,297	3,178	2,081	2,230	0,995	0,972**	0,633*
courier12	0,642	0,539	0,649	0,455	0,353**	0,383	0,192*
times12i	1,175	0,950	0,935	0,678	0,521	0,497**	0,266*
oldeng16	0,976	0,813	0,663	0,489	0,414**	0,489	0,244*
wingdings72-1	0,192	0,157**	0,215	0,215	0,222	0,200	0,103*
wingdings72-2	0,236	0,182**	0,431	0,202	0,200	0,206	0,095*
wingdings72-3	0,204	0,102**	0,489	0,205	0,192	0,276	0,071*
retangulos-size1	0,026*	0,026*	0,645	0,243	0,139	0,342	0,066**
retangulos-size2	0,143	0,155	0,673	0,242	0,139**	0,432	0,074*

CAPÍTULO 5

Reconhecimento de Forma com Codificação de Vizinhança

O reconhecimento de formas é um problema que tem grande importância no padrão de interface para descrição de conteúdo multimídia MPEG-7, no qual são incorporados descritores de forma. As bases de dados, os experimentos e os descritores de forma do MPEG-7 estão descritos na Seção 2.3. Esta seção propõe a utilização da codificação de vizinhança como descritor de forma e avalia o método proposto usando a base *Core Experiment CE-Shape-1* parte A2.

O método proposto neste capítulo extrai os códigos de vizinhança de uma imagem, então reduz o conjunto de códigos extraídos através do algoritmo de redução RED2 e calcula a similaridade entre duas imagens comparando seus conjuntos reduzidos códigos através de um algoritmo de *template matching*. Antes da extração dos códigos, a rotação de cada imagem é corrigida por um processo chamado padronização de rotação.

A Figura 5.1 mostra as três etapas utilizadas pelo método proposto para comparar duas formas. Inicialmente, a imagem é rotacioanda em torno do seu centro de massa, de modo que a direção de maior variância na imagem passe a ser a direção vertical, esse processo é chamado padronização de rotação. Em seguida, os códigos de vizinhança são extraídos da imagem, como descrito no Algoritmo 3.2. O conjunto de códigos extraídos é reduzido através do algoritmo de redução do conjunto de códigos RED2, Algoritmo 3.5. Para medir-se a distância entre dois conjunto de códigos reduzidos, cada um destes conjuntos é transformado numa cadeia ordenada pelos seus valores de centro x e y, nesta ordem. E essas cadeias são comparadas via *template matching*. O restante do capítulo explica o procedimento de padronização de rotação, como o *template matching* é implementado para comparar conjuntos de códigos de vizinhança e mostra os experimentos realizados.

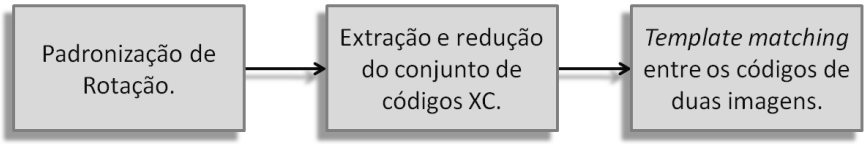


Figura 5.1 Diagrama de fluxo do método de reconhecimento forma usando codificação de vizinhança. A padronização de rotação rotaciona uma imagem para uma forma canônica. A extração de códigos é realizada gerando um código para cada pixel preto da imagem e o conjunto de códigos é reduzido pelo algoritmo RED2. A distância entre duas imagens é calculada comparando cadeias de códigos XC através de *template matching*.

5.1 Padronização de Rotação

A padronização de rotação é aplicada a cada imagem antes da extração dos códigos de vizinhança para limitar o número de possíveis rotações de uma imagem. A idéia dessa técnica é rotacionar uma imagem em torno do seu centro de massa de modo que, após a rotação, a imagem esteja numa forma canônica em relação à rotação.

Dada uma imagem original, pode-se gerar diversas imagens derivadas através de rotações em torno do centro de massa da imagem. A padronização de rotação é uma transformação aplicada à imagem original ou a qualquer uma das imagens derivadas que gera, para cada imagem, uma mesma resposta, que é chamada imagem canônica ou imagem padronizada em rotação. A imagem canônica de uma forma é obtida a partir de qualquer rotação desta em torno do seu centro de massa.

A técnica utilizada para gerar a padronização de rotação é a análise dos componentes principais [Smi02, Shl05, Shl09]. Esta técnica é capaz de identificar a direção de maior variância na imagem. E permite realizar uma transformação nesta imagem de modo que a direção de maior variância passe a ser a direção vertical.

A padronização de rotação utiliza a posição de cada pixel preto da imagem para calcular: o centro de massa, a direção de maior variância e a matriz de padronização da imagem. Então cada posição dos pixels pretos da imagem é transformada através da matriz de padronização. Com as novas posições dos pixels pretos é possível reconstruir a imagem padronizada em ro-

tação. O processo de padronização é descrito como segue:

1. **Transladar todos os pontos para o centro de massa.** O centro de massa de uma imagem binária é definido como a média das posições dos seus pixels pretos. Esta etapa calcula o centro de massa e subtrai esse valor de cada posição de pixel preto, ao final os pontos ficarão centrados em torno do ponto (0,0). Essa operação garante que a rotação vai ocorrer em torno do centro de massa, que passa a ser o ponto (0,0).

   Seja A uma matriz $2 \times m$ onde cada coluna é a posição (x_i, y_i), $1 \leqslant i \leqslant m$, de cada um dos m pixels pretos de uma imagem binária P:

   $$A = \begin{bmatrix} x_1 & x_2 & \dots & x_m \\ y_1 & y_2 & \dots & y_m \end{bmatrix}.$$

   O centro de massa $\mathbf{c}$ da imagem P é o vetor de média de todos os vetores que compõem a matriz A:

   $$\mathbf{c} = \begin{bmatrix} c_x \\ c_y \end{bmatrix} = \frac{1}{m} \sum_{i=1}^{m} \begin{bmatrix} x_i \\ y_i \end{bmatrix}.$$

   A matriz A_0 é a matriz A com todas as colunas subtraídas de $\mathbf{c}$,

   $$A_0 = \begin{bmatrix} x_1 - c_x & x_2 - c_x & \dots & x_m - c_x \\ y_1 - c_y & y_2 - c_y & \dots & y_m - c_y \end{bmatrix},$$

   A matriz A_0 contém a posição de todos os pontos da imagem P transladados para o centro de massa, dessa forma a média dos pontos de A_0 é o vetor $[0,0]^T$:

   $$\frac{1}{m} \sum_{i=1}^{m} \begin{bmatrix} x_i - c_x \\ y_i - c_y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

2. **Calcular a matriz de covariância dos pontos transformados.** Esta matriz de covariância tem dimensões 2×2, sua diagonal secundária é formada pela covariância C_{xy} entre os valores x e y das posições dos pontos da imagem, e a diagonal principal contém a variância C_{xx} dos valores de x e a variância C_{yy} dos valores de y.

A matriz de covariância C_{A_0} de A_0 pode ser calculada multiplicando a matriz A_0 por sua transposta A_0^T:

$$C_{A_0} = \frac{1}{m-1} A_0 A_0^T = \begin{bmatrix} C_{xx} & C_{xy} \\ C_{xy} & C_{yy} \end{bmatrix}.$$

3. **Calcular os autovalores e autovetores da matriz de covariância.** Os autovalores e autovetores podem ser calculados através da diagonalização da matriz de covariância [Kol98].

   O escalar λ é um autovalor da matriz C_{A_0} se existe um vetor não nulo $\mathbf{u}$ tal que $C_{A_0}\mathbf{u} = \lambda\mathbf{u}$, e todo vetor não nulo $\mathbf{u}$ que satisfaz esta condição é um autovetor de C_{A_0} associado ao autovalor λ.

   Os autovalores de C_{A_0} podem ser encontrados através da sua diagonalização. Para tanto é preciso resolver a equação característica:

$$det(\lambda I_2 - C_{A_0}) = 0,$$

   em que I_2 é a matriz identidade 2×2.

$$\begin{aligned} det(\lambda I_2 - C_{A_0}) &= det\left(\begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} - \begin{bmatrix} C_{xx} & C_{xy} \\ C_{xy} & C_{yy} \end{bmatrix}\right) \\ &= det\left(\begin{bmatrix} \lambda - C_{xx} & -C_{xy} \\ -C_{xy} & \lambda - C_{yy} \end{bmatrix}\right) \\ &= (\lambda - C_{xx})(\lambda - C_{yy}) - C_{xy}^2 \\ &= \lambda^2 - (C_{xx} + C_{yy})\lambda - C_{xy}^2 + C_{xx}C_{yy} \quad = 0 \end{aligned}$$

   Dessa forma, pode-se calcular os autovalores $\lambda_\mathbf{u}$ e $\lambda_\mathbf{v}$ de matriz C_{A_0} como:

$$\lambda_\mathbf{u} = \frac{C_{xx} + C_{yy} + \sqrt{\Delta}}{2};\ \lambda_\mathbf{v} = \frac{C_{xx} + C_{yy} - \sqrt{\Delta}}{2}, \Delta = (C_{xx} + C_{yy})^2 - 4(C_{xx}C_{yy} - C_{xy}^2).$$

Uma vez que $C_{xx} \geqslant 0$, $C_{yy} \geqslant 0$ e $\sqrt{\Delta} \geqslant 0$, $\lambda_{\mathbf{u}} \geqslant \lambda_{\mathbf{v}}$. Portanto λ_u é o autovalor associado ao autovetor $\mathbf{u}$ correspontende à maior variância na imagem.

Pode-se calcular os autovetores através da equação $C_{A_0}\mathbf{u} = \lambda\mathbf{u}$:

$$\begin{bmatrix} C_{xx} & C_{xy} \\ C_{xy} & C_{yy} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} = \lambda_{\mathbf{u}} \begin{bmatrix} u_x \\ u_y \end{bmatrix},$$

pela definição, um autovetor é todo vetor não nulo $\mathbf{u}$ que satisfaz a condição $C_{A_0}\mathbf{u} = \lambda\mathbf{u}$, o autovetor escolhido é aquele que tem $u_x = 1$, fazendo, pela equação acima, $u_y = \frac{\lambda_u - C_{xx}}{C_{xy}}$. Portanto os autovetores $\mathbf{u}$ e $\mathbf{v}$ de C_{A_0} são:

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} 1 \\ (\lambda_u - C_{xx})/C_{xy} \end{bmatrix}; \ \mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 1 \\ (\lambda_v - C_{xx})/C_{xy} \end{bmatrix}.$$

4. **Montar a matriz de padronização.** A matriz de padronização é formada pelos dois autovetores da matriz de covariância. Se a primeira coluna da matriz for formada pelo autovetor de maior autovalor, então, após a trasformação das posições dos pontos, a direção de maior variância vai ficar alinhada com o eixo horizontal da imagem. E se a primeira coluna da matriz for formada pelo autovetor de menor autovalor,então, após a trasformação das posições dos pontos, a direção de maior variância vai ficar alinhada com o eixo vertical da imagem. É importante que os autovetores que forma a matriz de transformação sejam normalizados (módulo do vetor igual a 1) para evitar que haja mudança de escala na transformação. A matriz de transformação T é

$$T = \begin{bmatrix} v_x/\sqrt{v_x^2 + v_y^2} & u_x/\sqrt{u_x^2 + u_y^2} \\ v_y/\sqrt{v_x^2 + v_y^2} & u_y/\sqrt{u_x^2 + u_y^2} \end{bmatrix}.$$

5. **Transformar os dados.** A transformação dos dados é realizada multiplicando cada ponto da imagem pela matriz de transformação. Depois cada ponto é transladado para o antigo centro de massa. A matriz com a posição de todos os pontos da imagem padronizada em rotação é R:

$$R = TA_0 + \begin{bmatrix} c_x & c_x & \dots & c_x \\ c_y & c_y & \dots & c_y \end{bmatrix}_{2\times m}.$$

Os resultados da padronização de rotação podem ser observados na Figura 5.2. Nesta figura estão imagens de duas classes da base *Core Experiment CE-Shape-1* A2 (Seção 2.3). Cada classe tem 6 imagens, uma imagem original e 5 derivadas através de rotações de 9^o, 36^o, 45^o (uma rotação de 9^o seguida de outra rotação de 36^o), 90^o e 150^o. A figura mostra o resultado da padronização para cada imagem. Nota-se que, para a linha de cima da figura, a operação funcionou como esperado para as 5 primeiras imagens, porém a última imagem apresenta um espelhamento em relação ao eixo horizontal. Para a linha de baixo da figura nota-se que também foram gerados dois tipos de imagens onde um tipo é uma versão do outro rotacionado em 180^o.

Os problemas de espelhamento e rotação em 180^o limitam as intenções da padronização de rotação. Estes problemas ocorrem devido aos sentido dos autovetores que formam a matriz de transformação. Para uso prático do método de padronização de rotação precisa-se assumir que quatro imagens canônicas podem ser geradas ao invés de apenas uma, pois duas imagens canônicas geradas a partir de rotações diferentes de uma mesma imagem podem estar: (1) idênticas, se os sentidos de ambos os autovetores coincidirem; (2) espelhadas em relação ao eixo horizontal, devido a uma inversão no sentido do autovetor de menor autovalor; (3) rotacionada em 180^o, devido a uma inversão no sentido do autovetor de maior autovalor; (4) ou apresentar as duas últimas variações ao mesmo tempo.

5.2 Distância entre Conjuntos de Códigos

Para calcular a dissimilaridade entre dois conjuntos de códigos XC, cada conjunto é organizado em uma cadeia ordenada pelos seus valores dos centros (x,y). Então a distância entre duas cadeias é calculada usando uma versão adaptada da distância de edição (*edit distance* [TK06]). A distância de edição utiliza a técnica de *template matching* para calcular o menor custo para se converter uma cadeia de teste em uma cadeia de referência. A conversão de uma cadeia em outra se dá através da inserção, remoção ou substituição de elementos. A distância de edição associa cada um dos m elementos de uma cadeia de referência com cada um dos n elementos de uma cadeia de teste através de uma matriz $m \times n$.

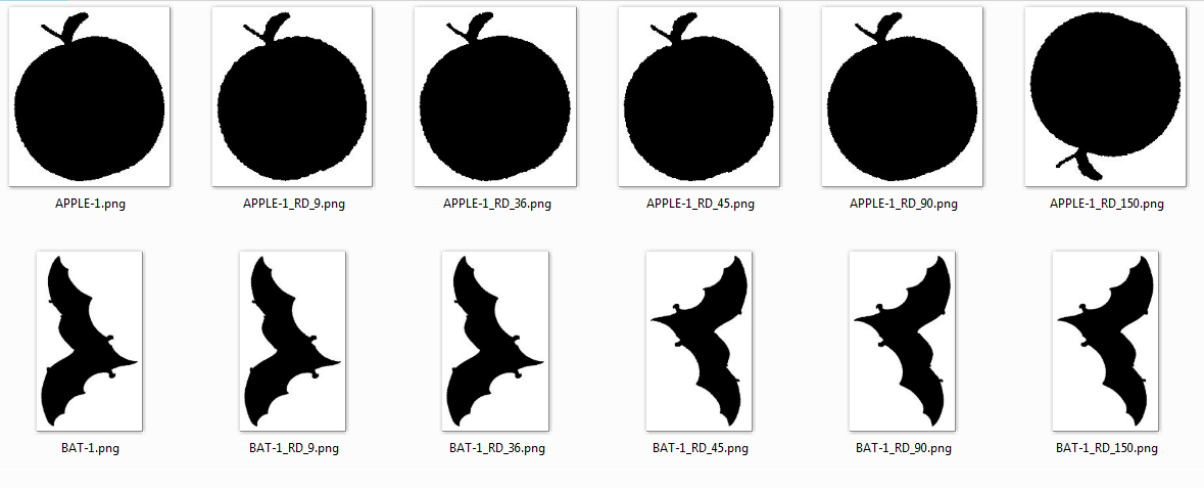


Figura 5.2 Exemplos da padronização de rotação. Em cada linha as imagens estavam originalmente rotacionadas em 0, 9^o, 36^o, 45^o, 90^o e 150^o. Na primeira linha ocorreu um problema de espelhamento da imagem canônica em relação ao eixo horizontal. E na segunda linha metade das imagens canônicas apareceram rotacionadas em 180 graus.

A Figura 5.3 mostra um exemplo da distância de edição para cadeias de caracteres. A linha desenhada na matriz indica o caminho percorrido para calcular a menor distância. Em (a) a cadeia de teste é transformada na cadeia de referência através da inserção de um símbolo, a inserção identificada como uma linha horizontal no caminho da menor distância. Em (b) a cadeia de teste é transformada na cadeia de referência através da substituição de um símbolo, que é indicada como um círculo no caminho da menor distância. Em (c) a cadeia de teste é transformada na cadeia de referência através da remoção de um símbolo, a remoção é descrita como uma linha vertical no caminho da menor distância. Em (d) as cadeias são idênticas, não precisam de inserção, remoção ou substituição para se tornarem iguais, portanto a distância entre elas é zero.

Os custos para inserção, remoção e substituição de códigos de vizinhança forem definidos empiricamente. O custo de se inserir um código XC da conFiguração torre φ_r da cadeia de referência na cadeia de teste é $d(i, j|i-1, j)$:

$$d(i, j|i-1, j) = n_r^2 + l_r^2 + s_r^2 + o_r^2.$$

O custo de se remover um código XC da conFiguração torre φ_t da cadeia de teste é $d(i, j|i, j-$

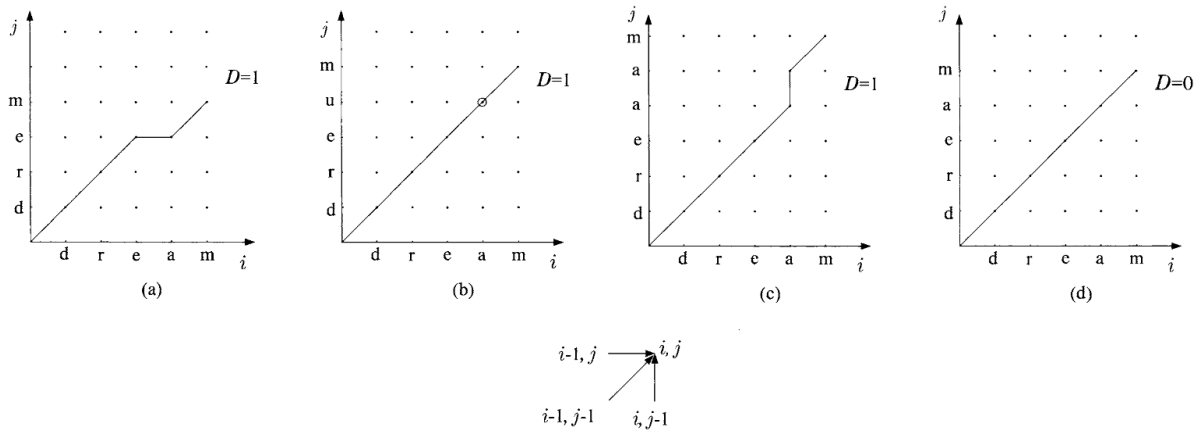


Figura 5.3 Cálculo da distância de edição para cadeias de caracteres. O custo de inserção, remoção e substituição é igual a 1 para qualquer caractere: (a) uma inserção; (b) uma substituição; (c) uma remoção; (d) duas cadeias idênticas não precisam de inserção, remoção ou substituição para se tornarem iguais, portanto a distância entre elas é zero. Figura retirada de [TK06].

1):

$$d(i,j|i,j-1) = n_t^2 + l_t^2 + s_t^2 + o_t^2.$$

E o custo de substituir um código XC da conFiguração torre φ_t da cadeia de teste por um código XC da conFiguração torre φ_r da cadeia de referência é $d(i,j|i-2,j-1)$:

$$d(i,j|i-1,j-1) = 2\times(x_t-x_r)^2 + 2\times(y_t-y_r)^2 + (n_t-n_r)^2 + (l_t-l_r)^2 + (s_t-s_r)^2 + (o_t-o_r)^2.$$

O Algoritmo 5.1 mostra como calcular a distância de edição entre duas cadeias de códigos de vizinhança. Este algorimo calcula o custo mínimo global de converter uma cadeia em outra minimizando o custo local, escolhendo sempre pela tarefa menos custosa entre substituição, remoção e inserção (linha 15). O algoritmo recebe como entrada duas cadeias de códigos XC cujos códigos estão ordenados pela posição dos centros, (x,y): uma cadeia de referência com m códigos e uma cadeia de teste com n códigos. E da como saída a distância de edição entre as duas cadeias. O primeiro passo do algortimo é criar uma matriz $(m+1)\times(n+1)$, $D = [d_{ij}]$ (linha 1), essa matriz é usada para calcular o custo de transformação da cadeia de teste na cadeia de referência. Na linha 2, a posição d_{00} é inicializada com zero, essa posição não indica qualquer relação entre as cadeias. Nas linhas de 3 a 5, é calculado o custo de inserir cada

elemento do padrão de referência na cadeia de teste. Nas linhas de 6 a 8, é calculado o custo de remover cada elemento do padrão de teste. Nas linhas de 9 a 17 é calculado efetimavente o menor custo de casamento entre cada elemento das duas cadeias. Na linha 11, é calculado o custo c_1 de se chegar à posição d_{rt} da matriz partindo da posição $d_{(r-1)(t-1)}$, para tanto é preciso calcular o custo de substituição do t-ésimo elemento da cadeia de teste pelo r-ésimo elemento da cadeia de referência. Na linha 12, é calculado o custo c_2 de se chegar à posição d_{rt} da matriz partindo da posição $d_{r(t-1)}$, para tanto é calculado o custo de se remover o t-ésimo elemento da cadeia de teste. Na linha 13, é calculado o custo c_3 de se chegar à posição d_{rt} da matriz partindo da posição $d_{(r-1)t}$, para tanto é calculado o custo de se inserir o r-ésimo elemento da cadeia de referência na cadeia de teste. O custo de se chegar à posição d_{rt} é o menor dos três custos calculados (linha 15). A distância de edição é o custo de se chegar à posição d_{mn} da matriz, então esse valor é retornado como resultado (linha 18).

5.3 Distância entre Duas Formas

Para medir a distância entre uma imagem de forma A e outra imagem de forma B o seguinte procedimento é realizado:

1. ambas as imagens são padromizadas quanto à rotação;
2. para a imagem B, após a padronização de rotação, são geradas mais 3 imagens através de espelhamento, rotação em 180 graus e espelhamento seguido de rotação 1m 180 graus;
3. o conjunto de códigos de cada uma das cinco imagens é extraído e reduzido via algoritmo de redução RED2 (Algoritmo 3.5);
4. a distância de edição é calculada entre a imagem A padronizada de rotação e cada uma das quatro versões da imagem B;
5. a distância entre a imagem A e a imagem B é a menor das quatro distâncias calculadas.

Algoritmo 5.1: Calculando a distância entre duas cadeias de códigos XC.

Entrada: Duas cadeias de códigos XC cujos códigos estão ordenados pela posição dos centros (x,y): uma cadeia de referência com m códigos e uma cadeia de teste com n códigos.

Saída: A distância de edição entre as duas cadeias

crie uma matriz $(m+1)\times(n+1)$, $D=[d_{ij}]$;
$d_{00}=0$;
para $r=1,\ldots,m$ **faça**
    $d_{r0}=d_{(r-1)0}+n_r^2+l_r^2+s_r^2+o_r^2$;
fim
para $t=1,\ldots,n$ **faça**
    $d_{0t}=d_{0(t-1)}+n_t^2+l_t^2+s_t^2+o_t^2$;
fim
para $r=1,\ldots,m$ **faça**
    para $t=1,\ldots,n$ **faça**
        $c_1=d_{(r-1)(t-1)}+2\times(x_t-x_r)^2+2\times(y_t-y_r)^2+(n_t-n_r)^2+(l_t-l_r)^2$
            $+(s_t-s_r)^2+(o_t-o_r)^2$;
        $c_2=d_{r(t-1)}+n_t^2+l_t^2+s_t^2+o_t^2$;
        $c_3=d_{(r-1)t}+n_r^2+l_r^2+s_r^2+o_r^2$;
        $d_{rt}=min(c_1,c_2,c_3)$
    fim
fim
retorna d_{mn}

5.4 Experimentos

O experimento foi realizados usando o teste padrão para a base *Core Experiment CE-Shape-1* A2, como está descrito na Seção 2.3, e compara os resultados com os métodos P298, P320, P517, P567 e P687, descritos na mesma seção. A base de imagens A2 avalia a robustez do descritor a variações de rotação. Esta base contém 420 formas. São 70 formas apresentas em seis rotações diferentes cada. As rotações são: 0^o, 9^o, 36^o, 45^o (uma rotação de 9^o seguida de outra rotação de 36^o), 90^o e 150^o. Exemplos de imagens da base A2 podem ser visualizadas na Figura 2.10. O método de teste padrão para a base A2 é recuperar, para cada uma das 420 imagens, as 6 imagens com menor distância. Conta-se um acerto cada vez que uma das seis imagem da mesma classe é recuperada, portanto o número máximo de acertos é $6 \times 420 = 2520$. E o acerto percentual é

$$100 \times \textit{número de acertos}/2520.$$

A distância usada entre duas imagens de forma é calculada como especificado na seção anterior. Os resultados do experimentos estão expostos na Tabela 5.1. Percebe-se que o método proposto apresentou um resultado tão bom quanto os outros métodos. A taxa de acerto de 99,52% indica que apenas duas das 420 imagens da base de teste foram incorretamente classificadas. Apesar de não se ter amostras suficientes para se fazer um teste de hipótese, é aceitável assumir que não há diferença entre a taxa de acerto do método proposto e as maiores taxas de acerto da mesma tabela.

Tabela 5.1 Taxas de acerto no reconhecimento de forma utilizando codificação de vizinhança (método proposto), sob a base *Core Experiment CE-Shape-1* A2, comparados com os algoritmos P298, P320, P517, P567 e P687, descritos na Seção 2.3.

Base	P298	P320	P517	P567	P687	Proposto
A2	100,0	99,37	100,0	97,46	99,60	99,52

CAPÍTULO 6

Conclusão e Trabalhos Futuros

O presente trabalho apresentou a codificação de vizinhança como um método para representar imagens binárias extendendo a codificação da forma como foi proposta originalmente em [TTD99, TT06b, TT06a]. A maneira como essa codificação é abordada nesta dissertação permite reconstruir a imagem completamente sem perdas. Permite também usar novas funções de vizinhança. Algumas novas funções de vizinhança são propostas além daquelas usadas originalmente, ainda é possível definir novas funções de vizinhança. Foram propostos três novos algoritmos para a redução do conjunto de códigos de vizinhança. O conjunto de códigos reduzidos gerado por cada um dos algoritmos propostos permite reconstruir a imagem sem perdas. Também foi proposto um método de compressão de imagens binárias usando codificação de vizinhança, o método proposto conseguiu superar a taxa de compressão de alguns dos principais formatos de compressão de imagens sem perda. A codificação de vizinhança também foi avaliada como descritor de forma e avaliada para o problema de reconhecimento de formas rotacionadas na base *Core Experiment CE-Shape-1* A2. Como pré-processamento na tarefa de reconhecimento, foi proposto o método de padronização de rotação, uma transformação que deixa toda forma numa rotação canônica. Algumas das técnicas propostas foram descritas num artigo [dCTT+10] aceito para a conferência *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

As contribuições desta dissertação podem ser expressas resumidamente na lista:

- a formalização da codificação de vizinhança;
- a possibilidade de construir novas funções de vizinhança;
- os algoritmos para a redução do conjunto de códigos, cujo conjunto de códigos selecionados permite reconstruir a imagem sem perdas;

- o método de compressão de imagens usando a codificação de vizinhança;
- o método de padronização de rotação de formas;
- o descritor de formas usando codificação de vizinhança.

Essas contribuições abrem novos problemas, os quais são propostos como trabalhos futuros e estão descritos na lista a seguir.

- Análise da complexidade algoritmica de processamento e memória dos métodos propostos.
- Propõe-se criar uma taxonomia de imagens binárias que leva em consideração a dispersão dos pixels pretos na imagem, algumas possíveis classes desta taxonomia são: *halftone*, desenho, texto, linhas finas, linhas grossas. Um conjunto de funções braço pode ser definido para representar cada tipo de imagem da taxonomia, dessa forma espera-se alcançar uma maior redução no conjunto de códigos necessário para representar a imagem sem perdas e um aumento na taxa de compressão usando codificação de vizinhança. Uma possível forma de definir essas funções braços é através de um algoritmo genético.
- Testar novas configurações de funções braço. Um estudo preliminar aponta ganho na compressão usando codificação de vizinhança usando diferentes configuração de funções braço para uma mesma imagem [Ten09].
- Os métodos propostos para a redução do conjunto de códigos não são ótimos, espera-se encontrar um algoritmo que seja ótimo e computacionalmente pouco custoso para essa tarefa.
- Segundo [Hol75] um algoritmo genético é capaz de encontrar a resposta ótima para um problema, dado tempo suficiente, portanto acredita-se ser possível reformular o algoritmo genético para redução do conjunto de códigos de forma a alcançar sempre a redução ótima.
- Espera-se também que seja possível encontrar uma forma de estimar o número mínimo de códigos de vizinhança necessários para representar uma imagem. Essa estimativa pode

ser usada como critério de parada para o algoritmo genético para redução do conjunto de códigos.

- Os algoritmos para redução do conjunto de códigos leva em consideração apenas o número de códigos gerados, contudo, para a tarefa de compressão os algoritmos para a redução do conjunto de códigos devem levar em consideração a entropia do conjunto reduzido de códigos.

- A padronização de rotação proposta não apresentou os resultados esperados, uma análise mais apurada sobre a matriz de transformação pode fazer com que a padronização de rotação gere apenas uma imagem canônica em vez de quatro como foi empregado. Se, ao invés de transformar os dados usando a matriz com os autovetores, se usasse uma matriz de rotação para rotacionar a direção de maior variância de modo que ela coincida com a direção vertical, o número de imagens canônicas seria reduzido para dois. E essas duas imagens canônicas estariam rotacionadas em 180 graus uma em relação a outra, e poderiam ser reduzidas para apenas uma imagem canônica empregando uma heurística como. Um exemplo de heurística para esse problema é dividir a imagem em duas subimagens (lado direito e esquerdo da linha vertical que divide a imagem ao meio), então gerara a versão espelhada da imagem da esquerda, depois eliminar os pixels pretos de cada subimagem que estão em posições equivalentes, se a subimagem que permanescer com mais pixels pretos for a subimagem de esquerda, a imagem é rotacionada em 180 graus alcançando a forma canônica.

- O método de compressão usando codificação de vizinhança mostrou-se mais adequado para imagens com grandes áreas contínuas, portanto deve ser avaliado em outras bases que possuam imagens com essa característica. Uma base de imagens com essa característica e um método novo que está sendo empregado para compressão desse tipo de imagem é proposto em [ACNCRD09, AC08, SCBRD07, SCRD05].

- Para atenuar as limitações da compressão com código de vizinhança em imagens grandes, sugere-se o uso de técnicas de análise multiresolução, com por exemplo, pirâmides

de imagem [GW10]. Cada pixel nas subimagens de menor ressolução numa pirâmide de imagens representa vários pixels numa subimagem de maior resolução na mesma pirâmide. Se a codificação de vizinhaça for empregada nas imagens de menor resolução, cada código de vizinhança representará mais pixels numa subimagem de mairo ressolução.

- Em [TT06b, TT06a] foram propostos operadores de vizinhança sobre os códigos de vizinhança que apresentam funcionamento semelhante a operadores morfológicos. Essa é um tópico que pode ser explorado com mais profundidade.

- Uma vez que um algoritmo genético pode ser bastante custoso em tempo, é possível propor um formato de arquivo para compressão que seja capaz de ser otimizado com o tempo. Através de um algoritmo genético pode-se definir um método que tenta sempre reduzir o tamanho da imagem comprimida com codificação de vizinhança, fazendo uso da capacidade de processamento ociosa de um computador. Ou mesmo, imagens que possuem partes idênticas, podem trocar informação para reconhecer que possuem a mesma informação e uma pode usar a representação da outra, se essa for mais compacta.

- Estender a codificação de vizinhança para imagens em tons de cinza e imagens multiespectrais. Uma proposta para usar codificação de vizinhança em imagens em tons de cinza é: realizar segmentação da imagem, representar cada segmentos usando codificação de vizinhança, armazenar a textura de cada segmento.

- A compressão usando codificação de vizinhança supera a compressão MH do padrão CCITT Group 3, que é empregada até hoje para a transmissão de fax via rede telefônica. Esse padrão já é empregado há 40 anos e ainda não foi substituído porque utiliza correção de erros. O codificação de vizinhança é redundante, o que facilita a criação de correção de erros para essa codificação. Uma proposta de trabalho futuro é avaliar a relevância de um novo padrão para compressão de imagens binária, resiliente a erros de transmissão, que pode ser usado para transmissão de fax em redes analógicas.

A codificação de vizinhança é uma técnica de representação de imagens extremamente nova

e que possui diversos aspectos a serem desenvolvidos. Possivelmente, a maior contribuição desta dissertação seja o grande número de problemas que ela levanta enquanto define uma nova abordagem para imagens digitais.

Referências Bibliográficas

[AC08] Sergio Alcaraz-Corona. *Image Compression Estimating the Markov Order of Dependencies*. PhD thesis, Instituto Tecnologico y de Estudios Superiores de Monterrey, December 2008.

[ACNCRD09] Sergio Alcaraz-Corona, Ricardo A. Neri-Calderón, and Ramón M. Rodríguez-Dagnino. Efficient bilevel image compression by grouping symbols of chain coding techniques. *Optical Engineering*, 48(3), March 2009.

[Bar06] Mauro Barni. *Document And Image Compression*. CRC Press, 1st. edition, May 2006.

[Bob01] M. Bober. Mpeg-7 visual shape descriptors. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):716 – 719, 2001.

[BPA00] M. Bober, W. Price, and J. Atkinson. The contour shape descriptor for mpeg-7 and its applications. *ICCE. 2000 Digest of Technical Papers. International Conference on Consumer Electronics*, pages 286 – 287, 2000.

[BYL+10] X. Bai, X. Yang, L. Latecki, W. Liu, and Z. Tu. Learning context-sensitive shape similarity by graph transduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[CK96] G. Chuang and C. C. Kuo. Wavelet descriptor of planar curves: theory and aplications. *IEEE Tranasactions on Image Processing*, 5:56–70, 1996.

[dA07] Carlos Wilson Dantas de Almeida. Recuperação de imagens baseada em uma abordagem híbrida. Master's thesis, Universidade Federal de Pernambuco (UFPE) - Centro de Informática (CIn), 2007.

[dC07] Tiago Buarque Assunção de Carvalho. Um estudo sobre funções de distância aplicadas a algoritmos de aprendizagem de máquina. Trabalho de Graduação em Ciências da Computação - Universidade Federal de Pernambuco / Centro de Informática, agosto 2007. http://www.cin.ufpe.br/ tg/2007-1/tbac.pdf.

[dC08] Tiago Buarque Assunção de Carvalho. Normalização de rotação em imagens binárias, uma proposta. Agosto 2008.

[dCTT+10] Tiago Buarque Assunção de Carvalho, D. J. Tenório, I. R. Tsang, G. D. C. Cavalcanti, and I. J. Tsang. Neighborhood coding for bilevel image compression and shape recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2010.

[DJLW08] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2), April 2008. Article 5.

[DLR09] Mark S. Drewa, Tim K. Lee, and Andrew Rova. Shape retrieval with eigen css search. *Image and Vision Computing*, 27:748–755, 2009.

[Dou94] Edward R. Dougherty. *Digital image processing methods*. CRC Press, January 1994. ISBN-13: 9780824789275, ISBN-10: 082478927X.

[ES03] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 1st edition edition, 2003. ISBN: 3-540-40184-9.

[GA96] J. Gailly and M. Adler. *RFC1951 - DEFLATE Compressed Data Format Specification version 1.3*. Aladdin Enterprises, May 1996.

[GA10] J. Gailly and M. Adler. Compression algorithm (deflate), 2010. http://www.gzip.org/algorithm.txt.

[Gal04] B. Galwas. E-book for fundamentals of microwaves. In *Microwaves, Radar and Wireless Communications, 2004. MIKON-2004. 15th International Conference on Volume: 3*, pages 1071 – 1075. IEEE, 2004.

[Gla07] Henry M. Gladney. *Preserving Digital Information*. Springer-Verlag Berlin Heidelberg, 2007. ISBN 978-3-540-37886-0.

[GW10] Rafael C. Gonzalez and Richard E. Woods. *Processamento digital de imagens*. PEARSON, 3 edition, 2010. ISBN-13: 9788576054016.

[Hol75] John H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.

[Huf52] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E (Institute of Radio Engineers)*, pages 1098–1101, September 1952.

[Ima] ImageMagick. Imagemagick® is a software suite to create, edit, and compose bitmap images. http://www.imagemagick.org/.

[IT88] (CCITT) ITU-T. *Recommendation T.6. Facsimile Coding Schemes And Coding Control Functions For Group 4 Facsimile Apparatus*, 1988.

[IT03] (CCITT) ITU-T. *Recommendation T.4. Standardization of Group 3 Facsimile Terminals for Document Transmission*, july 2003.

[JBB+98] Corinne Le Buhan Jordan, S. Bhattacharjee, F. Bossen, F. Jordan, and T. Ebrahimi. Shape representation and coding of visual objects in multimedia applications - an overview. *Annals of Telecommunications*, 53(5-6):164–178, 1998. ISSN 0003-4347.

[JK10] Yabing Jiang and Evangelos Katsamakas. The impact of e-book technology on book retailing. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1 – 8. IEEE, 2010. Digital Object Identifier: 10.1109/HICSS.2010.383.

[KH90] A. Khotanzan and Y. H. Hong. Invariant image recognition by Zernike moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:489–497, 1990.

[Kol98] Bernard Kolman. *Introdução à Álgebra Linear com Aplicações*. LTC – Livros Técnicos e Científicos Editora S.A., 6 edition, 1998.

[LA00] Daniel L. Lau and Gonzalo R. Arce. *Modern Digital Halftoning*. Marcel Dekker, 2000. ISBN 0-8247-0456-8.

[Les05] Michael Lesk. *Understanding Digital Libraries*. The Morgan Kaufmann Series in Multimedia Information and Systems. Morgan Kaufmann Publishers is an imprint of Elsevier, 2 edition, 2005. ISBN: 1-55860-924-5.

[LL00] L. J. Latecki and R. Lakämper. Shape similarity measure based on correspondence of visual parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1185–1190, 2000.

[LLE00] Longin Jan Latecki, Rolf Lakämper, and Ulrich Eckhardt. Shape descriptors for non-rigid shapes with a single closed contour. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 424–429, 2000.

[LR01] Subhash R. Lele and Joan T. Richtsmeier. *An invariant approach to analysis of shapes*. Interdisciplinary Statistic Series. Chapman & Hall / CRC, 2001. ISBN 0-8493-0319-2.

[LZ09] Binli Li and Yinwei Zhan. A new and efficient descriptor for region based shape retrieval. In *WRI World Congress on Computer Science and Information Engineering*, pages 71 – 75, 2009.

[Mar04] José M. Martínez. *ISO/IEC JTC1/SC29/WG11N6828 MPEG-7 Overview (version 10)*, October 2004.

[Mey00] Paul L. Meyer. *Probabilidade - Aplicações à Estatística*. LTC, 2 ed. edition, 2000. ISBN: 9788521602941.

[MKP02] José M. Martínez, Rob Koenen, and Fernando Pereira. MPEG-7: the generic multimedia content description standard, part 1. *IEEE Multimedia*, 9(2):78 – 87, 2002.

[MMS00] Stpéphane Marchand-Maillet and Yazid M. Sharaiha. *Binary Digital Image Processing - A Discrete Approach*. Academic Press, 2000. ISBN 0-12-470505-7.

[MR96] J.D. Murray and W. Van Ryper. *Encyclopedia of Graphics File Formats*. O'Reilly & Associates, 2nd edition edition, 1996. http://www.fileformat.info/mirror/egff/.

[MV02] Murat Mese and P. P. Vaidyanathan. Recent advances in digital halftoning and inverse halftoning methods. *IEEE Transactions On Circuits And Systems I*, 49(6):790 – 805, June 2002.

[NG08] A. Noorhidawati and F. Gibb. How students use e-books - reading or referring. *Malaysian Journal of Library and Information Science*, 13(2):1 – 14, 2008.

[PA09] Glauber M. Pires and Aluizio F. R. Araujo. A stochastic neural model for fast classification of binary images. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2547–2552, June 2009.

[Pai10] Thaysa Suely Beltrao Paiva. Capítulo 1 - bibliotecas digitais. Technical report, Universidade Federal de Pernambuco (UFPE) - Centro de Informática (CIn), março 2010. <tsbp2@cin.ufpe.br>.

[Par02] Alvaro Pardo. Lossless shape representation and coding. Technical report, Facultad de Ingeniería, Universidad de la República, C.C. 30, Montevideo, Uruguay, April 2002.

[PDF08] D. Petreus, I. Dobra, and C. Farcas. E-book for evaluating core loss, different technique and materials. In IEEE, editor, *Electronics System-Integration Technology Conference*, pages 1069 – 1074, 2008.

[RMB97] M. M. Reid, R. J. Millar, and N. D. Black. Second-generation image coding: An overview. *ACM Computing Surveys*, 29(1), 1997.

[Ros98] Kenneth Rosen. *Discrete Mathematics and its Applications*. McGraw-Hill Higher Education, 4th ed. edition, 1998. ISBN: 0-07-289905-0.

[Sam04] Hanan Samet. Object-based and image-based object representation. *ACM Computing Surveys*, 36(2):159–217, June 2004.

[SCBRD07] Hermilo Sánchez-Cruz, Ernesto Bribiesca, and Ramón M. Rodríguez-Dagninoc. Efficiency of chain codes to represent binary objects. *Pattern Recognition*, 40:1660–1674, 2007.

[SCRD05] Hermilo Sánchez-Cruz and Ramón M. Rodríguez-Dagnino. Compressing bilevel images by means of a three-bit chain code. *Optical Engineering*, 44(9), September 2005.

[Shl05] Jonathon Shlens. A tutorial on principal component analysis. Technical report, Systems Neurobiology Laboratory, Salk Insitute for Biological Studies, December 2005. http://www.cs.cmu.edu/ elaw/papers/pca.pdf.

[Shl09] Jonathon Shlens. A tutorial on principal component analysis. Technical report, Center for Neural Science, New York University andSystems Neurobiology Laboratory, Salk Insitute for Biological Studies, April 2009. http://www.snl.salk.edu/ shlens/pca.pdf.

[Smi02] Lindsay I Smith. A tutorial on principal components analysis. Technical report, University of Otago - Department of Computer Science, February 2002.

[Ten09] Denise Jaeger Tenório. Xc4 - compressão usando códigos de vizinhança. Technical report, Universidade Federal de Pernambuco (UFPE) - Centro de Informática (CIn), novembro 2009.

[TK06] S. Theodoridis and K. Koutroumbas. *Pattern Recognition, 3rd Edition*. Academic Press, 2006.

[TT06a] I. R. Tsang and I. J. Tsang. *Image Analysis and Recognition*, volume 4141/2006 of *Lecture Notes in Computer Science*, chapter Pattern Recognition Using

Neighborhood Coding, pages I: 600–611. Springer Berlin / Heidelberg, September 2006.

[TT06b] I. R. Tsang and I. J. Tsang. Neighbourhood vector as shape parameter for pattern recognition. In *International Joint Conference on Neural Networks (IJCNN)*, pages 3204–3209, July 2006.

[TTD99] I. J. Tsang, I. R. Tsang, and D. Van Dyck. Image coding using neighbourhood relations. *Pattern Recognition Letters*, 20:1279–1286, 1999.

[TTD00] I. J. Tsang, I. R. Tsang, and D. Van Dyck. Cluster diversity and entropy on the percolation model: The lattice animal identification algorithm. *Physical Review E - Statistical Physics, Plasmas, Fluids and Related Interdisciplinary Topics*, 62(5):6004–6014, 2000.

[WM97] D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *J. Artificial Intelligence Research*, 6:1 – 34, 1997.

[WMB99] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing gigabytes: compressing and indexing documents and images*. The Morgan Kaufman Series in Multimedia Information and Systems. Morgan Kaufman Publishers, 2nd ed. edition, 1999. ISBN 1-55860-570-3.

[WNC87] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[Zib01] Carla Raquel Faria Lopes Zibreira. Descrição e procura de vídeo baseadas na forma. Master's thesis, Image Group - Instituto de Telecomunicações, Instituto Superior Técnico, Av. Rovisco Pais - 1049-001 Lisboa, PORTUGAL, Julho 2001.

[ZL77] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337 – 343, 1977.