Operadores Morfológicos Baseados em
Grafos de Vizinhaça
Uma Extensão da “*MMach Toolbox*”

Francisco de Assis Zampirolli

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA OBTENÇÃO DO GRAU DE MESTRE
EM
MATEMÁTICA APLICADA

Área de Concentração : Ciência da Computação
Orientador : Prof. Dr. Junior Barrera

- São Paulo, 10 de Abril de 1997-

Operadores Morfológicos Baseados em
Grafos de Vizinhaça
Uma Extensão da “*MMach Toolbox*”

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e apresentada por Francisco de Assis Zampiroli e aprovada
pela Comissão Julgadora.

São Paulo, 10 de Abril de 1997

Banca Examinadora :

- Prof. Dr. Junior Barrera (orientador) - IME-USP
- Prof. Dr. Carlos Eduardo Ferreira - IME-USP
- Prof. Dr. Roberto de Alencar Lotufo - DCA/FEE-UNICAMP

à minha família,

Agradecimentos

Ao Prof. Dr. **Junior Barrera** pela orientação, apoio, incentivo e confiança durante a realização desse trabalho,

Aos Profs. Drs. **Roberto de Alencar Lotufo** e **Routo Terada** pela interessante oportunidade de trabalharmos em equipe, e ao Prof. Dr. **Carlos Eduardo Ferreira** pelas críticas para o enriquecimento desse texto,

Aos amigos **Roberto Hirata Jr.** e **Nina Sumiko Tomita** pela companhia no estudo e nos trabalhos em equipe, e à **Guillermo Pablo Salas** pelo companherismo,

À **Ayumi Kato** e ao **Claus Akira Matsushigue** pelas críticas em parte deste texto,

Ao **CNPq** e à **Olivetti do Brasil** pelo apoio financeiro recebido durante a elaboração desse trabalho,

A todos que direta ou indiretamente colaboraram desde o início do mestrado,

À minha família principalmente aos meus pais que sempre confiaram e apoiaram meus estudos,

o meu muito obrigado.

Resumo

A Morfologia Matemática estuda os mapeamentos, também chamados operadores, entre reticulados completos. Através da teoria da Morfologia Matemática pode-se provar que qualquer operador pode ser decomposto em termos de quatro classes de operadores elementares: dilatação, erosão, anti-dilatação e anti-erosão. Outro importante resultado é que os operadores elementares são caracterizados por funções estruturantes em reticulados superados. Uma importante família de funções estruturantes são as baseadas em grafos de vizinhança.

Podemos construir uma linguagem formal sobre um conjunto finito de reticulados completos usando os operadores elementares da Morfologia Matemática e as operações de ínfimo e supremo sobre cada reticulado completo. Esta linguagem é chamada linguagem morfológica, uma implementação dessa linguagem é denominada máquina morfológica. Um exemplo de máquina morfológica é a "*MMach Toolbox*", implementada no sistema de processamento de imagens "KHOROS".

Em [Vin89] são estudados operadores morfológicos construídos a partir de grafos de vizinhança. Inspirado neste trabalho propomos uma extensão da "*MMach Toolbox*" que dispõe de operadores baseados em grafos de vizinhança. Com estes operadores podemos abordar problemas de processamento de imagens que dependem de relações de vizinhança entre objetos, tais como: cálculo do cisalhamento de um material metálico; segmentação da imagem da polpa do eucalipto; análise reticular da imagem celular endotelial da córnea; segmentação de imagens em geral.

Abstract

Mathematical Morphology studies the mappings, also called operators, between complete lattices. Through Mathematical Morphology theory it is possible to prove that any operator can be decomposed in terms of four classes of elementary operators: dilation, erosion, anti-dilation and anti-erosion. Other important result is that in sup-generated lattices these elementary operators are characterized by structuring functions. Examples of structuring functions are the ones based on neighborhood graphs.

We can construct a formal language on a finite set of complete lattices, using the elementary operators of Mathematical Morphology and the operations of infimum and supremum of each lattice. This language is called a morphological language. An implementation of the morphological language is called morphological machine. An example of a morphological machine is the “*MMach Toolbox*”, implemented on the image processing system “KHOROS”.

In [Vin89] are studied morphological operators constructed through neighborhood graphs. Inspired in this paper we proposed an extension of “*MMach Toolbox*” that deals with operators based on neighborhood graphs. Applying these operators we can solve image processing problems which depend on neighborhood relations between objects, for example: detection of defect lines in alloy images; segmentation of eucalypt pulpwood images; network analysis of cornea endothelial cell images; segmentation of images in general.

Índice

Lista de Figuras	v
Lista de Tabelas	xi
Capítulo 1. Introdução	1
1.1. Processamento de imagens	1
1.2. Morfologia Matemática	5
1.3. A “ <i>MMach Toolbox</i> ” antiga	6
1.4. Extensão da “ <i>MMach Toolbox</i> ”	6
1.5. Organização do texto	8
Capítulo 2. Noções Preliminares	11
2.1. Elementos de Teoria dos Grafos	11
2.2. Elementos de Geometria Computacional	14
2.3. Construção de grafos de vizinhança	16
2.4. Imagens partição e imagem rotulada	22
Capítulo 3. Operadores morfológicos	25

3.1.	Caracterização dos operadores elementares	25
3.2.	Aplicação em imagens binárias	29
3.3.	Aplicação em imagens em níveis de cinza	31
Capítulo 4.	Extensão da “ <i>MMach Toolbox</i> ”	35
4.1.	Plataforma de trabalho KHOROS	35
4.2.	Arquitetura da “ <i>MMach Toolbox</i> ”	37
4.3.	A “ <i>MMach Toolbox</i> ” estendida	38
4.4.	Implementações básicas	39
Capítulo 5.	Algoritmos para a extensão da “ <i>MMach Toolbox</i> ”	47
5.1.	Grafos de vizinhança	47
5.2.	Dilatação e Erosão	51
5.3.	Função distância	53
5.4.	Função distância geodésica	56
5.5.	Rotulações	57
5.6.	Zonas de influência	61
5.7.	Junção de regiões	64
5.8.	Zonas flats	66
5.9.	Filtro por área	67
Capítulo 6.	Aplicações da “ <i>MMach Toolbox</i> ” estendida	71
6.1.	Material metálico	71
6.2.	Eucalipto	76
6.3.	Córnea	80
6.4.	Segmentações	84
6.5.	Músculo	89

Capítulo 7. Conclusão	95
Apêndice A. Estrutura de dados das funções estruturantes	99
Apêndice B. Algoritmo para a triangulação de Delaunay	101
B.1. Algoritmo	101
B.2. Corretude e desempenho	103
Apêndice C. Operadores da “ <i>MMach Toolbox</i> ”	111
Apêndice D. Documentações para a “ <i>MMach Toolbox</i> ” estendida	117
D.1. Páginas na Internet	117
D.2. Ajuda “ <i>on line</i> ” do KHOROS	119
Bibliografia	121
Índice Remissivo	123

Lista de Figuras

1.1	(a) Imagem sintética de uma liga metálica e (b) Marcadores.	2
1.2	Imagem com uma possível linha de cisalhamento.	2
1.3	Imagem da polpa do eucalipto.	3
1.4	Área improdutiva da polpa do eucalipto em destaque.	3
1.5	(a) Imagem do bloco (b) Marcadores.	4
1.6	Segmentação do bloco.	4
1.7	Imagem microscópica do músculo humano.	4
1.8	Imagem segmentada.	5
1.9	Diagrama da “ <i>MMach Toolbox</i> ” antiga.	7
1.10	Diagrama da “ <i>MMach Toolbox</i> ” estendida.	8
2.1	Grafos quase regulares.	13
2.2	Diagramas de Voronoi ($\cdots$) e Delaunay ($-$) ([JRZ91]).	17
2.3	Grafo de Gabriel.	19
2.4	Grafo de vizinhança relativa.	19

2.5	Árvore geradora mínima.	20
2.6	Distância máxima d .	20
2.7	Grafo dos k vizinhos mais próximos onde $k = 1$.	21
2.8	Vizinhos do baricentro.	21
2.9	(a) Imagem partição e (b) Imagem partição com linha divisória.	22
2.10	Imagem rotulada.	23
3.1	Função estruturante com propriedade de dilatação.	28
3.2	Dilatação binária com função estruturante q.i.t..	30
3.3	Dilatação binária com função estruturante baseada em grafos.	30
3.4	Dilatação de uma imagem em níveis de cinza, representada por função, por uma função estruturante i.t..	32
3.5	Dilatação de uma imagem em níveis de cinza, representada por função, por uma função estruturante q.i.t..	32
4.1	Janela do CANTATA mostrando um workspace.	36
4.2	Dilatação em imagem implícita.	38
4.3	Dilatação em imagem explícita (baseada em grafos).	45
5.1	Dados de entrada e saída do programa vdil visualizados pelo programa vexpslide.	52
5.2	Estrutura de grafos armazenada em <i>graph</i> e visualizada pelo programa vgslice.	52
5.3	Dados de entrada e saída do programa vero visualizados pelo programa vexpslide.	53
5.4	Dados de entrada e saída do programa vdist visualizados pelo programa vexpslide.	55

5.5	Dados de entrada e saída do programa <code>vgeodist</code> visualizados pelos programa <code>vexpslide</code> .	56
5.6	Dados de entrada e saída do programa <code>vlabel</code> visualizados pelos programa <code>vexpslide</code> , usando a rotulação binária.	59
5.7	Dados de entrada e saída do programa <code>vwatersh</code> visualizados pelo programa <code>vexpslide</code> .	63
5.8	Dados de entrada e saída do programa <code>vmosaic</code> visualizados pelos programa <code>vexpslide</code> .	66
5.9	Workspace visualizando os programas contidos em <code>vflats</code> .	68
5.10	Workspace visualizando os programas contidos em <code>vfilterarea</code> .	69
6.1	Imagem sintética de um material metálico.	72
6.2	(a) Esqueleto e (b) Marcadores.	72
6.3	(a) Supremo da imagem I com a Figura 6.2.a e (b) Grafo de vizinhança.	73
6.4	Distância em grafos com relação aos marcadores.	74
6.5	Imagem com uma possível linha de cisalhamento.	74
6.6	Workspace para encontrar cisalhamento em material metálico.	75
6.7	Imagem da polpa do eucalipto.	76
6.8	Segmentação usando “ <i>top hat</i> ” seguido da abertura por reconstrução.	76
6.9	Imagem mostrando o supremo da imagem I com a Figura 6.8.	77
6.10	Grafo de vizinhança marcando vértices com arestas maiores.	78
6.11	Fechamento seguido de dilatação baseados em grafos.	78
6.12	Resultado da segmentação da imagem da polpa do eucalipto.	78
6.13	Workspace para segmentação da polpa do eucalipto.	79
6.14	Duas imagens ilustrando partições contendo células da córnea. O nível de cinza corresponde a área da partição.	80

6.15	O nível de cinza das células corresponde ao número de vizinhos.	81
6.16	Granulometria usando fechamento incremental.	81
6.17	Histograma da granulometria.	82
6.18	Workspace para análise de células da córnea.	83
6.19	Imagem do bloco.	84
6.20	(a) Filtro alternado seqüencial e (b) Rotulação.	85
6.21	(a) Marcadores a mão e (b) Zonas flats marcadas.	85
6.22	Segmentação do bloco.	85
6.23	Workspace para segmentação usando o programa vmosaic.	86
6.24	Imagem do fotógrafo.	87
6.25	(a) Filtro alternado seqüencial e (b) Rotulação.	87
6.26	(a) Marcadores e (b) Zonas flats marcadas.	88
6.27	Segmentação do fotógrafo.	88
6.28	Imagem microscópica do músculo humano.	89
6.29	(a) Segmentação da Figura 6.28 e (b) Supremo de I com a imagem à esquerda com níveis de cinza menores para melhor visualização.	89
6.30	(a) Medidas regionais e (b) Grafo de vizinhança.	90
6.31	(a) “ <i>Threshad</i> ” da Figura 6.30.a e (b) Máximos locais da Figura 6.30.a.	91
6.32	(a) “ <i>Threshad</i> ” da Figura 6.31.b e (b) Subtração da Figura 6.31.a com a Figura 6.32.a.	91
6.33	(a) Dilatação da Figura 6.32.b e (b) Negação da imagem à esquerda.	91
6.34	(a) Ínfimo da Figura 6.31.a com a Figura 6.33.b e (b) Outra forma de ver a imagem à esquerda.	92
6.35	Segmentação das células do músculo humano.	93

6.36	Workspace visualizando os passos para a segmentação das células do músculo.	93
6.37	Visualização do procedimento process do workspace anterior.	94
7.1	Interface gráfica do KHOROS versão 2.1 - CANTATA	97
B.1	Ilustração de um passo do algoritmo da triangulação de Delaunay.	103
B.2	Ilustração do Lema B.2.	104
B.3	Ilustração do Lema B.3.	105
B.4	Ilustração do Teorema B.4.	105
B.5	Ilustração do Teorema B.5.	106
B.6	Ilustração do Lema B.6.	107
B.7	Ilustração do Lema B.6.	107
B.8	Ilustração do Lema B.6.	108
B.9	Ilustração do Lema B.9.	109
B.10	Ilustração da demonstração do Teorema B.11.	110
D.1	Página na Internet.	118
D.2	Ajuda <i>"on line"</i> do KHOROS.	119

Lista de Tabelas

C.1	Nível “tools”.	111
C.2	Nível “random”.	111
C.3	Nível medidas.	112
C.4	Nível funções estruturantes.	112
C.5	Nível imagens explícitas.	112
C.6	Operadores básicos de imagens.	113
C.7	Primeiro nível de operadores.	114
C.8	Segundo nível de operadores.	115
C.9	Terceiro nível de operadores.	116

CAPÍTULO 1

Introdução

Apresentamos neste capítulo uma introdução do nosso trabalho sobre operadores morfológicos baseados em grafos de vizinhança. Começamos descrevendo alguns problemas de processamento de imagens que podem ser resolvidos através destes operadores. Em seguida, apresentamos a Morfologia Matemática, teoria que descreve matematicamente estes operadores. Apresentamos também a “MMach Toolbox”, ferramenta que viabiliza a solução de problemas de processamento de imagens usando os operadores morfológicos. Finalmente, apresentamos a organização desse texto.

1.1. Processamento de imagens

Pense numa imagem. Pode ser uma imagem vista de uma nave espacial ou de um microscópio eletrônico, ou até mesmo uma imagem criada pela nossa imaginação, como uma pintura. Será que é possível extrair automaticamente informações quantitativas dessas imagens? Após o advento da aplicação dos computadores digitais ao processamento de imagens, a resposta a esta pergunta é cada vez mais categoricamente positiva.

Podemos exemplificar a resposta a esta pergunta ilustrando alguns problemas resolvidos por processamento de imagens.

Considere uma imagem microscópica de uma liga metálica, como ilustrado na Figura 1.1.a. É possível encontrar uma linha de cisalhamento entre dois pontos pré-determinados? Veja Figura 1.1.b.

FIGURA 1.1. (a) Imagem sintética de uma liga metálica e (b) Marcadores.

A resposta a esta pergunta é afirmativa. Podemos encontrar a solução usando dois conceitos. Primeiro, podemos encontrar a solução manipulando os *pixels* (do inglês “*picture element*”, menor elemento da imagem) das imagens apresentadas na Figura 1.1. O resultado pode ser visto na Figura 1.2. Uma outra solução é manipular objetos (componentes conexas) de imagens. Podemos considerar cada buraco da Figura 1.1.a como sendo um objeto. É possível também usar informações sobre as vizinhanças entre estes objetos. Manipulando estes objetos com as informações de vizinhança, também encontramos a mesma solução.

FIGURA 1.2. Imagem com uma possível linha de cisalhamento.

Outro exemplo ilustrando problemas de processamento de imagens é encontrar a área improdutiva da polpa do eucalipto, na industrialização da celulose. Veja na Figura 1.3 uma imagem microscópica da polpa do eucalipto.

FIGURA 1.3. Imagem da polpa do eucalipto.

Podemos também usar os dois conceitos apresentados anteriormente, pixel e objeto, para resolver este problema. Veja a solução na Figura 1.4.

FIGURA 1.4. Área improdutiva da polpa do eucalipto em destaque.

Na Figura 1.5.a mostramos uma imagem de um bloco. É possível segmentar ([GW92]) as faces deste bloco e o fundo desta image, preservando seus limites? Marcando estas faces, como ilustra a Figura 1.5.b, podemos encontrar a imagem da Figura 1.6. Se resolvermos este problema usando objetos, podemos propagar as regiões usando a noção de máxima proximidade de regiões, onde a vizinhança é determinada pela adjacência destas regiões.

Na imagem microscópica do músculo humano, ilustrada na Figura 1.7, desejamos encontrar as células claras rodeadas de células escuras. Como resolver este problema?

FIGURA 1.5. (a) Imagem do bloco (b) Marcadores.

FIGURA 1.6. Segmentação do bloco.

Podemos considerar que cada célula é um objeto e usar uma relação de vizinhança determinada pela adjacência entre células. Manipulando estas informações encontramos a solução ilustrada na Figura 1.8.

FIGURA 1.7. Imagem microscópica do músculo humano.

Podemos armazenar os objetos, citados nos problemas anteriores, em vetores contendo as coordenadas espaciais do centro geométrico dos objetos. Podemos também armazenar

FIGURA 1.8. Imagem segmentada.

neste vetor alguma informação relevante aos objetos. Como, por exemplo, média dos níveis de cinza, área, etc. Essas funções (*imagens explícitas*) são transformadas tendo em vista relações de vizinhança entre objetos, representadas por grafos de vizinhança.

Em certas aplicações de Processamento de imagens é mais natural usarmos o conceito de objetos do que o de pixels. Nos exemplos apresentados anteriormente, podemos encontrar a solução desejada facilmente, como veremos no Capítulo 6.

Além disso, comparando os dois conceitos, vê-se que podemos resolver certos problemas com menor esforço computacional usando o conceito de objeto, pois não é mais necessário percorrer a imagem, mas sim uma transformação desta contendo apenas informações relevantes.

1.2. Morfologia Matemática

Uma forma elegante de resolver problemas de processamento de imagens é utilizar uma base teórica consistente. Uma destas teorias é a *Morfologia Matemática*, criada na década de 60 por Jean Serra e George Matheron na *École Nationale Supérieure des Mines de Paris*, em *Fontainebleau, França*. Nesta teoria estudamos transformações entre reticulados completos. Uma transformação entre reticulados completos é chamada *operador morfológico*. Na Morfologia Matemática temos quatro classes de operadores básicos, chamados *elementares*. Podemos provar que todos os operadores morfológicos podem ser obtidos a partir de combinações destes operadores elementares ([BB93]).

Um outro resultado importante da Morfologia Matemática (quando os reticulados considerados são sup-geradores) é que esses operadores são caracterizados por *funções*

estruturantes^[i] ([BB93]). O que é uma função estruturante? Os operadores elementares têm como principal característica percorrer todos os pixels de uma imagem e para cada pixel visitar seus pixels “vizinhos” (determinados por uma função estruturante) fazendo uma conta e armazenando o resultado neste pixel visitado.

Usando os operadores morfológicos podemos construir uma *linguagem formal* ([BB92]). Uma implementação de uma linguagem formal é uma *máquina morfológica*. Um exemplo de máquina morfológica é a “*MMach Toolbox*”([BB94] e [BBL]).

1.3. A “*MMach Toolbox*” antiga

A “*MMach Toolbox*” é uma caixa de ferramentas para resolver problemas de processamento de imagens usando operadores morfológicos ([BBL]).

Uma forma tradicional de usar os operadores morfológicos na “*MMach Toolbox*” é considerar uma imagem armazenada em forma de matriz, onde cada pixel da imagem corresponde a uma coordenada da matriz. Esta imagem é chamada *imagem implícita*. A função estruturante é também armazenada em forma de matriz. Este exemplo de função estruturante ilustra o conceito de vizinha entre pixels. Na Figura 1.9 apresentamos este caso onde temos uma imagem sintética microscópica de uma liga metálica como entrada e uma função estruturante particular caracterizada por um conjunto chamado *elemento estruturante* ([BB94]). A saída também é uma imagem implícita ilustrando uma possível linha de cisalhamento.

1.4. Extensão da “*MMach Toolbox*”

O problema do cisalhamento, apresentado na seção anterior, pode também ser resolvido usando um conceito de vizinhança diferente, a vizinhança entre objetos.

O primeiro passo para isto é transformar a imagem implícita em *imagem explícita*, imagem contendo pixels esparsos de uma imagem implícita, armazenando as coordenadas e um valor para cada pixel. A partir deste ponto, os elementos de uma imagem explícita serão chamados de *objetos*. As coordenadas dos pixels da imagem explícita correspondem ao baricentro (centro geométrico) dos objetos da imagem implícita. Os valores dos pixels

^[i] As definições de reticulados, sup-geradores e funções estruturantes serão definidos no próximo capítulo.

FIGURA 1.9. Diagrama da “*MMach Toolbox*” antiga.

são determinados por *arquivos de medidas*, arquivos criados a partir de imagens implícitas ou de grafo de vizinhança.

Nos arquivos de medidas, cada *componente conexa* da imagem implícita (*imagens partição* ou *imagem rotulada*) produz uma lista de medidas. Por exemplo, podemos calcular, para cada objeto, sua área, perímetro, projeções horizontal e vertical, etc. Estas medidas são denominadas *medidas individuais*. Podemos também calcular medidas sobre nível de cinza. Neste caso, temos duas imagens implícitas de entrada, uma em níveis de cinza e a outra determinando os objetos (*imagens partição* ou *imagem rotulada*). Para cada objeto calculamos média, mediana, entropia e valor máximo dos níveis de cinza, armazenando estes cálculos num arquivo de medidas. Denominamos estas medidas por *medidas regionais*.

O próximo passo é construir *grafos de vizinhança*. Temos três formas de construção de grafos. A partir de uma imagem explícita e de uma imagem partição, construímos um grafo onde os vértices são determinados pela imagem explícita. As arestas são determinadas pelos objetos da imagem partição, onde existe uma aresta no grafo se seus extremos correspondem a partições (componentes conexas) adjacentes na imagem partição. Uma outra forma de criar grafos de vizinhança é usando uma imagem explícita fornecendo os vértices, as arestas são as com distância menor que um certo escalar. Podemos também construir grafos de vizinhança onde o número de vizinhos de cada vértice são os k mais próximos, onde k é um escalar.

A partir das imagens explícitas e dos grafos construídos podemos aplicar os operadores morfológicos da “*MMach Toolbox*”.

Finalmente, para visualizar as imagens explícitas e os grafos criamos duas ferramentas que têm como saída imagens implícitas.

Veja na Figura 1.10 uma ilustração de operadores morfológicos baseados em grafos de vizinhança.

FIGURA 1.10. Diagrama da “*MMach Toolbox*” estendida.

Quando executamos um operador morfológico sobre uma imagem explícita, usando uma função estruturante baseada em grafos de vizinhança, percorremos os objetos desta imagem e para cada objeto visitamos seus pixels “vizinhos” determinados pela função estruturante (grafo).

1.5. Organização do texto

Apresentamos no Capítulo 2 noções preliminares de teoria dos grafos e de geometria computacional. Mostramos também formas de construir grafos de vizinhança, dado um conjunto de pontos no plano. No Capítulo 3 mostramos resultados de teoria dos conjuntos, reticulados e Morfologia Matemática. No Capítulo 4 apresentamos como foi estendida a “*MMach Toolbox*”, discutimos ainda a plataforma de trabalho KHOROS e também a arquitetura e o polimorfismo da “*MMach Toolbox*”. No Capítulo 5 apresentamos vários algoritmos baseados em grafos de vizinhança. No Capítulo 6 apresentamos algumas aplicações usando a “*MMach Toolbox*” estendida. A conclusão é dada no Capítulo 7. Temos também quatro apêndices. No Apêndice A apresentamos a estrutura de dados para as

funções estruturantes. No Apêndice B aprofundamos o problema da triangulação de Delaunay, dando um algoritmo eficiente com sua corretude e seu desempenho. No Apêndice C apresentamos todos os operadores da “*MMach Toolbox*”. Finalmente, no Apêndice D apresentamos os dois tipos de documentação existentes para a “*MMach Toolbox*”.

... e a possibilidade de uma nova abordagem da história da literatura portuguesa, que não se limite a uma simples descrição dos fatos, mas que tente explicar os processos que os geraram. A obra de Almeida Faria é, portanto, uma obra de grande importância para a história da literatura portuguesa, e que merece ser lida e estudada por todos os interessados no assunto.

CAPÍTULO 2

Noções Preliminares

Este capítulo contém noções preliminares de Teoria dos Grafos e de Geometria Computacional, além de apresentar alguns tipos de construir grafos de vizinhança.

Um dos objetivos deste texto é resolver problemas práticos de processamento de imagens usando grafos de vizinhança. O primeiro passo para isto é estabelecer formas de construir grafos de vizinhança a partir de objetos de imagens explícitas (neste capítulo usaremos o conceito de pontos no plano para representar este objetos). Isto será feito na Seção 2.3, destacando a triangulação de Delaunay. Na Seção 2.4 definimos imagens partição e imagem rotulada.

2.1. Elementos de Teoria dos Grafos

Nesta seção apresentaremos definições de Teoria dos Grafos ([BM76] e [Bar92]).

A *Teoria dos Grafos* estuda a interrelação entre os elementos de um conjunto V . Um elemento de V é denominado *vértice*. A interrelação entre os vértices é representada por um conjunto A de pares de vértices, isto é, $A \subseteq V^2$. Um elemento de A é denominado *aresta*. Um *grafo* $G(V, A)$ é definido como uma estrutura constituída por um conjunto de vértices V e um conjunto de arestas A . Se o conjunto de vértices e o conjunto de arestas a que estamos nos referindo são óbvios pelo contexto o grafo $G(V, A)$ é denotado simplesmente por G .

Se u e v são os vértices extremos de uma aresta a então u é *adjacente* a v e a *incide* em u e v . A aresta a com extremos u e v também é denotada por uv .

Duas arestas com um extremo em comum são chamadas *adjacentes*. Duas arestas com os mesmos extremos são chamadas *paralelas ou múltiplas*. Uma aresta com extremos idênticos é chamada *laço*.

O *grau ou número de vizinhos* de um vértice v num grafo G , denotado por $g_G(v)$, é o número de arestas que incidem em v onde os laços são contados duas vezes.

Um grafo é *simples* se não tem laços e nem arestas múltiplas. Um grafo $G(V, A)$ é *finito* se V e A são ambos finitos.

Os nossos estudos se restringirão a grafos não orientados, simples e finitos.

Um *passeio* em um grafo é uma sequência finita e não vazia $P = \{v_0, a_1, v_1, \dots, a_k, v_k\}$, cujos termos são alternadamente vértices v_i e arestas a_j , e tal que, para todo i , $1 \leq i \leq k$, os extremos de a_i são v_{i-1} e v_i . Dizemos que P é um passeio de v_0 a v_k e que os vértices v_0 e v_k são a origem e o término de P , respectivamente. O inteiro k é o *comprimento* de P . Um *caminho* é um passeio sem vértices repetidos.

Um grafo G é *conexo* se para todo par de vértices distintos u, v em G existe um caminho de u para v . Dizemos que G é *desconexo* se G não for conexo.

Sejam u e v vértices de um grafo G . A *distância* de u a v em G , denotada por $d(u, v)$, é o comprimento de um caminho mais curto de u a v em G . Se não existe nenhum caminho de u a v então definimos $d(u, v)$ como sendo infinito, ou seja, $d(u, v) = \infty$.

Um grafo $G(V, A)$ é *k-regular* se $g_G(v) = k$ para todo $v \in V$. G é *regular* se é *k-regular* para algum k .

Um grafo $G'(V', A')$ é um *subgrafo* de um grafo $G(V, A)$, denotado por $G' \subseteq G$, se $V' \subseteq V$, $A' \subseteq A$ e para toda aresta de G' , seus extremos em G' são também seus extremos em G .

Seja $\mathbb{Z}$ o conjunto dos inteiros. Se $G'(V', A')$ é um subgrafo de um grafo regular $G(\mathbb{Z}^2, A)$ tal que $A' = A \cap V'^2$ então G' é dito um *grafo quase regular*. A Figura 2.1 mostra três exemplos de grafos quase regulares: em (a) temos uma *grade quadrada com vizinhança-4* ou *conectividade-4*; em (b) temos uma *grade quadrada com vizinhança-8* ou *conectividade-8*; e em (c) temos uma *grade hexagonal*.

FIGURA 2.1. Grafos quase regulares.

Se V é um conjunto de pontos do plano e A é um conjunto de arestas construídas pela análise das relações de vizinhança entre estes pontos então dizemos que $G(V, A)$ é um *grafo de vizinhança*.

Um grafo em *níveis de cinza* $G = (f, V, A)$ é a estrutura constituída pelo grafo $G(V, A)$ e pela função f de V em $K = \{0, 1, \dots, N - 1\}$ com $N \in \mathbb{Z}^+$. O grafo $G(V, A)$ é denominado *grafo de referência* de $G = (f, V, A)$.

Dois grafos em níveis de cinza G e G' são ditos *similares* se eles têm o mesmo grafo de referência, isto é, se $G(V, A) = G'(V', A')$.

O grafo G é chamado um *grafo binário* se $K = \{0, 1\}$. Os vértices v de G são chamados *vértices brancos* se $f(v) = 1$ e *vértices pretos* se $f(v) = 0$.

Dois *vértices* brancos v_1 e v_2 de um grafo binário G são ditos *conectados* se existe um caminho de v_1 a v_2 em G tal que todos os vértices deste caminho são vértices brancos. Um *vértice* v é dito *conectado a um conjunto* S de *vértices* brancos se v é conectado a algum vértice de S . Um *conjunto de vértices* brancos S é dito *conectado a um conjunto de vértices* brancos T se existe algum vértice em S conectado a algum vértice em T . Um *conjunto* S de *vértices* brancos é dito *conexo* se S não pode ser particionado em dois subconjuntos que não estão conectados um com outro. Uma *componente conexa* de um *conjunto de vértices* brancos S é um subconjunto conexo não vazio de S que não é conectado a nenhum outro vértice de S .

Podemos generalizar as definições de conectividade em grafos binários para grafos em níveis de cinza fazendo preto valer zero e branco valer um nível de cinza diferente de zero.

2.2. Elementos de Geometria Computacional

Nesta seção apresentaremos elementos de Geometria Computacional ([RS94]).

Geometria Computacional é a área da computação que trata de soluções de problemas geométricos. Como nosso trabalho estuda grafo de vizinhança é importante definirmos quais são as operações elementares da geometria computacional.

Inicialmente vamos escolher uma forma para representar as entidades geométricas. Usualmente representamos um *ponto* p no plano através de suas *coordenadas cartesianas* (x, y) , onde x e y são números reais ($x, y \in \mathbb{R}$) medidos a partir de alguma origem arbitrária ao longo de dois eixos ortogonais. A totalidade destes pontos constitui o *plano cartesiano* $\mathbb{R}^2$.

Uma *reta* no plano é definida por três coeficientes (A, B, C) . Um ponto (x, y) está na reta sse^[i] $Ax + By + C = 0$. Uma reta só está definida se $A \neq 0$ ou $B \neq 0$. Duas retas (A, B, C) e $(\alpha A, \alpha B, \alpha C)$ são representações da mesma reta, onde $\alpha \in \mathbb{R}$. As retas (A, B, C) e $(-A, -B, -C)$ são *opostas*.

Uma reta divide o plano em dois lados. Um ponto $p = (x, y)$ está no lado positivo da reta (A, B, C) se $Ax + By + C > 0$. Se $Ax + By + C < 0$ então p está no lado negativo.

O *ponto médio* m de dois pontos $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$ é definido por

$$m = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right).$$

Dados dois pontos $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$, a reta que passa por eles é definida por

$$\det \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x & y \end{pmatrix} = 0$$

onde $\det(X)$ é o determinante da matriz X .

Dados três pontos $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ e $p_3 = (x_3, y_3)$, seja

$$D = \det \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix}.$$

^[i] Lê-se: se e somente se.

Se $D > 0$ então p_3 está à direita da reta definida por p_1 e p_2 . Se $D = 0$ então p_1, p_2, p_3 são *colineares*. Se $D < 0$ então p_3 está à esquerda da reta definida por p_1 e p_2 .

Dados quatro pontos não colineares $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, $p_3 = (x_3, y_3)$ e $p_4 = (x_4, y_4)$, seja

$$D = \det \begin{pmatrix} 1 & x_1 & y_1 & x_1^2 + y_1^2 \\ 1 & x_2 & y_2 & x_2^2 + y_2^2 \\ 1 & x_3 & y_3 & x_3^2 + y_3^2 \\ 1 & x_4 & y_4 & x_4^2 + y_4^2 \end{pmatrix}.$$

Se $D < 0$ então p_4 pertence ao círculo definido por p_1, p_2, p_3 . Se $D > 0$ então p_4 não pertence ao círculo definido por p_1, p_2, p_3 .

O *segmento*^[ii] com extremidades p_1 e p_2 , denotado por p_1p_2 , é o conjunto dos pontos obtidos por combinação convexa de p_1 e p_2 , isto é,

$$p_1p_2 = \{p : p = \alpha p_1 + (1 - \alpha)p_2 \quad 0 \leq \alpha \leq 1\}.$$

Uma *semi-reta* é um segmento de reta com um dos extremos no infinito.

Uma função $d : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$ é chamada de *distância* se satisfaz as seguintes propriedades:

- (i) $d(p, p) = 0$, para todo $p \in \mathbb{R}^2$;
- (ii) $d(p, q) = d(q, p)$, para todo $p, q \in \mathbb{R}^2$;
- (iii) $d(p, q) > 0$, se $p \neq q$, para todo $p, q \in \mathbb{R}^2$;
- (iv) $d(p, q) + d(q, r) \geq d(p, r)$, para todo $p, q, r \in \mathbb{R}^2$.

Dados $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$ pontos do plano, definimos as seguintes distância entre p_1 e p_2 :

- (a) **Euclidiana:** $d_e(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$;
- (b) **City-Block:** $d_1(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$;
- (c) **Chessboard:** $d_\infty(p_1, p_2) = \max\{|x_1 - x_2|, |y_1 - y_2|\}$.

^[ii] Nesse texto iremos representar as arestas de um grafo de vizinhança por segmentos de retas.

Seja $\mathbb{Z}$ o conjunto dos inteiros. Seja x um elemento de $\mathbb{Z}^2$ e r um inteiro positivo. Um *disco digital* com distância d de centro x e raio r é um subconjunto de $\mathbb{Z}^2$ dado por $D_d(x, r) = \{y \in \mathbb{Z}^2 : d(x, y) \leq r\}$. A distância entre um ponto x e um subconjunto X de $\mathbb{Z}^2$ com distância d é um valor dado por $d(x, X) = \min\{d(x, y) : y \in X\}$.

O *triângulo* pqr determinado pelos pontos p, q e r é o conjunto de todas as combinações convexas dos três pontos, isto é,

$$pqr = \{s : s = \alpha_1 p + \alpha_2 q + \alpha_3 r, \sum \alpha_i = 1, \alpha_i \geq 0, i = 1, 2, 3\}.$$

Os segmentos pq, qr e rp formam a *fronteira* do triângulo pqr . Os pontos que não estão na fronteira estão no interior ou no exterior do triângulo. Dizemos que um triângulo é *degenerado* se seu interior for vazio. Denotamos por $\widehat{prq}$ o ângulo entre pr e rq .

Definimos *fecho convexo* de um conjunto P de pontos no plano como sendo o menor conjunto convexo^[iii] que contém P .

2.3. Construção de grafos de vizinhança

Nesta seção iremos considerar P como sendo um conjunto finito com n pontos no plano e usaremos apenas distância euclidiana.

Apresentaremos a seguir alguns métodos para construir arestas a partir de P ([Vin89], [RS94], [FC91], [GS85], [CE85], [Lee82] e [JRZ91]).

O primeiro método a ser apresentado será a triangulação de Delaunay. Este método tem tempo de execução $\Theta(n \lg n)$ ^[iv] ([FC91]). Dada a triangulação de Delaunay de P , existem outros métodos de construção de arestas para P em tempo $O(n)$, como por exemplo, o grafo de vizinhança relativa, o grafo de Gabriel e a árvore geradora mínima ([Vin89] e [RS94]).

2.3.1. Triangulação de Delaunay

A triangulação de Delaunay determina uma partição do plano para o conjunto P onde cada *partição*, também chamada *face*, é um triângulo com seus extremos sendo vértices

^[iii] Veja a definição de conjunto convexo em [Roc70].

^[iv] Veja em [CLR90] a definição da notação Θ .

de P . A definição da triangulação de Delaunay é obtida a partir da definição do diagrama de Voronoi, ambas apresentadas a seguir.

Considere um ponto $p_i \in P$, definimos *região de Voronoi* de p_i , denotado por $Z(p_i)$, da seguinte forma:

$$Z(p_i) = \{p \in \mathbb{R}^2 : d_e(p, p_i) \leq d_e(p, p_j), \forall j = 1, 2, \dots, n\},$$

onde d_e é a distância Euclidiana.

FIGURA 2.2. Diagramas de Voronoi ($\cdots$) e Delaunay ($-$) ([JRZ91]).

Note que $\{Z(p_i), i = 1, 2, \dots, n\}$ particiona o plano em regiões convexas onde a interseção de dois convexas é um segmento de reta, uma semi-reta ou uma reta. Esta decomposição é chamada *diagrama de Voronoi*.

O algoritmo watershed ou zona de influência ([Vin89]) é uma aproximação do diagrama de Voronoi pois trabalha no plano finito e considera regiões ao invés de pontos. No watershed as regiões (componentes conexas de uma imagem implícita) vão aumentando proporcionalmente e simultaneamente até se tocarem, quando isto ocorre, os pontos que pertencem a duas ou mais regiões fazem parte da zona de influência dessas regiões. O diagrama de Voronoi de P também é a zona de influência de P se o crescimento das regiões centradas em P for infinito.

Dado o diagrama de Voronoi de P , o *diagrama de Delaunay* é o grafo não orientado $G(P, A)$ onde A é o conjunto de arestas definido por

$$A = \{p_i p_j : |Z(p_i) \cap Z(p_j)| > 1; i, j = 1, 2, \dots, n \text{ e } i \neq j\},$$

onde $|x|$ é a cardinalidade de x .

Dizer que $|Z(p_i) \cap Z(p_j)| > 1$ é equivalente a dizer que $Z(p_i) \cap Z(p_j)$ é uma reta, uma semi-reta ou um segmento de reta não degenerado^[v]. Um diagrama de Delaunay de P é chamado *triangulação de Delaunay* se todas as faces internas do diagrama são triângulos.

Teorema 2.1. *Dado P com no mínimo três pontos, toda triangulação do fecho convexo de P possui exatamente $2n - v - 2$ triângulos e $3n - v - 3$ arestas, onde v é o número de pontos de P que estão na fronteira do fecho convexo de P .* $\diamond$

Exemplo 2.2. Um exemplo trivial de triangulação é o triângulo. Nele temos três pontos extremos (n) e três pontos na fronteira (v). Assim, temos um triângulo com três arestas e o Teorema 2.1 se aplica. $\diamond$

Uma importante aplicação deste teorema é que o número de triângulos e o número de arestas numa triangulação com n pontos são respectivamente $O(n)$ ([FC91], p. 75 e 76). Através deste resultado vários problemas apresentados neste capítulo serão resolvidos em tempo linear, quando o dado de entrada for uma triangulação.

2.3.2. Grafos derivados da triangulação de Delaunay

Apresentaremos a seguir três classes de grafos que podem ser gerados a partir da triangulação de Delaunay.

Grafo de Gabriel

O *grafo de Gabriel* de P é o grafo $G(P, A)$, onde A é o conjunto de arestas formado pelos segmentos de retas pq , onde p e q são pontos de P e o círculo de diâmetro pq passando por p e q é livre de pontos em P , veja Figura 2.3 (a aresta tracejada não pertença ao grafo de Gabriel). Pode-se provar que o grafo de Gabriel é um subgrafo da triangulação de Delaunay. Usando o Lema B.2 podemos concluir que toda aresta do grafo de Gabriel também é aresta da triangulação de Delaunay. Logo podemos encontrar o grafo de Gabriel a partir da triangulação de Delaunay.

^[v] Não degenerado significa, no contexto, de comprimento diferente de zero.

FIGURA 2.3. Grafo de Gabriel.

Grafo de vizinhança relativa

Dois pontos p e q de P são extremos de uma aresta do *grafo de vizinhança relativa* se nenhum outro ponto de P está mais próximo de ambos do que eles estão próximos entre si, veja Figura 2.4. Formalmente, pq é uma aresta deste grafo sse $d(p, r) \geq d(p, q)$ ou $d(q, r) \geq d(p, q)$, $\forall r \in P \setminus \{p, q\}$. Observe que toda aresta deste grafo satisfaz a definição de aresta do grafo de Gabriel. Logo o grafo de vizinhança relativa é um subgrafo do grafo de Gabriel.

FIGURA 2.4. Grafo de vizinhança relativa.

Árvore geradora mínima

Um grafo $G(P, A)$ é uma *árvore geradora mínima* se, para quaisquer dois vértices em P existe e é único o caminho que liga estes vértices e a soma do comprimento (distância euclidiana) das arestas é mínimo, veja Figura 2.5. Por [Pre], Teorema 6.1, pode-se provar que este grafo é um subgrafo da triangulação de Delaunay.

FIGURA 2.5. Árvore geradora mínima.

2.3.3. Outras formas de construção de grafos

Apresentaremos a seguir três outros métodos para a construção de grafos que não dependem da triangulação de Delaunay.

Distância máxima

Usando a distância Euclidiana vamos definir um grafo $G(P, A)$, onde a aresta $pq \in A$ se a distância entre p e q não é maior que uma distância d , veja Figura 2.6. Chamamos este grafo de *grafo de distância máxima*.

FIGURA 2.6. Distância máxima d .

K vizinhos mais próximos

Para cada ponto $p \in P$, definimos uma aresta $pq \in A$ se $q \in P$ é um dos k vizinhos mais próximos de p , veja Figura 2.7. O grafo $G(P, A)$ é chamado *grafo dos k vizinhos mais próximos*.

FIGURA 2.7. Grafo dos k vizinhos mais próximos onde $k = 1$.

Vizinhos do Baricentro

Definimos o *grafo dos vizinhos do baricentro* $G(P, A)$ por: dado $p \in P$, as arestas adjacentes a p em A têm extremos $\{p_1, \dots, p_k\}$ se p está incluído no fecho convexo destes pontos, veja Figura 2.8. A desvantagem desta construção é que ela não produz necessariamente um grafo único ([Vin89]).

FIGURA 2.8. Vizinhos do baricentro.

Nesta seção foi apresentado várias formas de se construir grafos de vizinhança a partir de pontos no plano. No Capítulo 5 apresentaremos algoritmos para os problemas dos k vizinhos e distância máxima, ficando como sugestão a implementação dos outros métodos.

Apesar de não termos implementado a triangulação de Delaunay, no Apêndice B damos um algoritmo ótimo para este problema.

2.4. Imagens partição e imagem rotulada

Definimos a seguir alguns conceitos de imagens utilizadas neste texto.

Definimos *imagem partição* como sendo a divisão de uma imagem em partições de tal forma que a união de todas as partições é a imagem e a interseção de duas partições distintas é vazio. As partições são diferenciadas por níveis de cinza distintos. Cada elemento destas são também chamadas de componentes conexas. Veja um exemplo de imagem partição na Figura 2.9-a.

Usamos também uma definição aproximada de imagem partição, onde existe uma linha na interseção de duas partições quaisquer. Veja um exemplo na Figura 2.9-b.

(a)

(b)

FIGURA 2.9. (a) Imagem partição e (b) Imagem partição com linha divisória.

Uma outra imagem utilizada neste texto é a *imagem rotulada*, onde temos componentes conexas (objetos) distintas por níveis de cinza. Observe que uma imagem partição com linha divisória é uma imagem rotulada. Veja um exemplo de imagem rotulada na Figura 2.10.

FIGURA 2.10. Imagem rotulada.

CAPÍTULO 3

Operadores morfológicos

A *Morfologia Matemática* é uma teoria que estuda a decomposição de operadores entre reticulados completos em termos de quatro classes de operadores: as dilatações, as erosões, as anti-dilatações e as anti-erosões. Através destes operadores, chamados de *operadores elementares*, construímos qualquer outro operador. Outro importante resultado é que, quando os reticulados possuem uma família sup-geradora, estes operadores podem ser caracterizados por funções estruturantes ([BB91] e [BB93]).

A base teórica deste trabalho se concentra numa função estruturante particular, a baseada em grafos de vizinhança. Nela definimos a relação de vizinhança entre objetos no plano.

Na Seção 3.1 apresentaremos a dilatação e a erosão caracterizadas por funções estruturantes em reticulados completos sup-gerados. Na Seção 3.2 mostraremos exemplos de funções estruturantes no reticulado completo das imagens binárias. Na Seção 3.3 apresentaremos outros exemplos de funções estruturantes no reticulado completo das imagens em níveis de cinza.

3.1. Caracterização dos operadores elementares

Nesta seção apresentaremos os operadores morfológicos elementares da Morfologia Matemática caracterizados por funções estruturantes ([BB91], [BB93], [Bar92], [Vin89] e [Ser88]).

Sejam $\mathcal{L}_1$ e $\mathcal{L}_2$ dois conjuntos quaisquer não vazios. Um *operador* ψ entre $\mathcal{L}_1$ e $\mathcal{L}_2$ é definido como um mapeamento de $\mathcal{L}_1$ em $\mathcal{L}_2$, $\psi : \mathcal{L}_1 \rightarrow \mathcal{L}_2$. O conjunto de todos os operadores entre $\mathcal{L}_1$ e $\mathcal{L}_2$ é denotado $\mathcal{L}_2^{\mathcal{L}_1}$.

Exemplo 3.1. Seja E um retângulo finito de $\mathbb{Z}^2$, onde $\mathbb{Z}$ é o conjunto dos números inteiros, e $K = [k_0, k_1] \subset \mathbb{Z}^+$ é o conjunto finito dos níveis de cinza. Então K^E é o conjunto de todas possíveis *imagens em níveis de cinza*. $\diamond$

Um conjunto $\mathcal{L}$ estruturado por uma relação $\leq$ é um *conjunto parcialmente ordenado* ou *poset*, denotado $\mathcal{L} = (\mathcal{L}, \leq)$, se a relação $\leq$ sobre elementos de $\mathcal{L}$ for reflexiva, anti-simétrica e transitiva ([Bir67]). A relação de ordem inversa $\geq$ em $\mathcal{L}$ é definida por $X \geq Y$ sse $Y \leq X$, $\forall X, Y \in \mathcal{L}$. Diremos que $(\mathcal{L}, \geq)$ ou simplesmente $\tilde{\mathcal{L}}$ é o *poset dual* de $\mathcal{L}$.

Sejam $L, U, X \in \mathcal{L} \neq \emptyset$ e $\mathcal{X} \subset \mathcal{L}$, dizemos que:

- U é *limitante superior* de $\mathcal{X}$ se $\forall X \in \mathcal{X}$ temos $X \leq U$;
- L é *limitante inferior* de $\mathcal{X}$ se $\forall X \in \mathcal{X}$ temos $L \leq X$;
- O *supremo* de $\mathcal{X}$, se existir, é o menor dos limitantes superiores, denotado $\vee \mathcal{X}$, i.e., $\vee \mathcal{X}$ é limitante superior de $\mathcal{X}$ e $\forall U \in \mathcal{L}$ limitante superior de $\mathcal{X}$, temos $\vee \mathcal{X} \leq U$;
- O *ínfimo* de $\mathcal{X}$, se existir, é o maior dos limitantes inferiores, denotado $\wedge \mathcal{X}$, i.e., $\wedge \mathcal{X}$ é limitante inferior de $\mathcal{X}$ e $\forall L \in \mathcal{L}$ limitante inferior de $\mathcal{X}$, temos $\wedge \mathcal{X} \leq L$.

Um poset $\mathcal{L}$ é um *reticulado completo* se todo subconjunto não vazio $\mathcal{X}$ de $\mathcal{L}$ tem um supremo e um ínfimo.

Um subconjunto próprio l de um reticulado completo $\mathcal{L}$ é chamado uma *família superadora* se todo elemento de $\mathcal{L}$ pode ser escrito como o supremo de subconjuntos de l . Os elementos de l serão denotados pelas letras minúsculas x e y . Assim, fazendo $l(X) = \{x \in l : x \leq X\}$ então $X = \vee l(X)$, $\forall X \in \mathcal{L}$.

Exemplo 3.2. O conjunto $\mathcal{P}(E)$ de todos os subconjuntos do conjunto não vazio E , com a relação usual de inclusão $\subset$ entre subconjuntos, é um reticulado completo. As operações de ínfimo e supremo são, respectivamente, as operações de interseção e união entre subconjuntos. Uma família sup-geradora l de $\mathcal{P}(E)$ é a coleção dos conjuntos que têm um único ponto, i.e., $\{\{x\} : x \in E\}$.

Um elemento X de $\mathcal{P}(E)$, onde $(\mathcal{P}(E), \subseteq)$ é um poset, pode representar, por exemplo, uma *imagem binária* de E em $\{0, 1\}$: o valor do ponto é 1 se ele pertence a X e 0, caso contrário. $\diamond$

Exemplo 3.3. A coleção K^E com a relação de ordem parcial $\leq$ dada por $\forall f_1, f_2 \in K^E, f_1 \leq f_2 \Leftrightarrow f_1(x) \leq f_2(x), \forall x \in E$ é um reticulado completo. As operações de ínfimo e supremo são, respectivamente, $\forall \mathcal{X} \subset K^E$,

$$\begin{aligned} (\wedge \mathcal{X})(x) &= \min\{f(x) : f \in \mathcal{X}\} \quad (x \in E) \quad e \\ (\vee \mathcal{X})(x) &= \max\{f(x) : f \in \mathcal{X}\} \quad (x \in E). \end{aligned}$$

Uma família sup-geradora l de K^E é a coleção das funções que são diferentes de k_0 ($K = [k_0, k_1]$) em um único ponto, i.e., $\{f_{p,v} : (p, v) \in E \times K\}$, onde $\forall x \in E$,

$$f_{p,v}(x) = \begin{cases} v & \text{se } x = p \\ k_0 & \text{c.c.} \end{cases}$$

$\diamond$

Sejam $(\mathcal{L}_1, \leq_1)$ e $(\mathcal{L}_2, \leq_2)$ reticulados completos. O supremo e o ínfimo em $\mathcal{L}_1$ serão denotados, respectivamente, $\vee_1$ e $\wedge_1$. E analogamente para $\mathcal{L}_2$. Suponha $\psi \in \mathcal{L}_2^{\mathcal{L}_1}$.

Dizemos que:

- ψ é uma *dilatação* sse ψ comuta com o supremo, i.e., $\psi(\vee_1 \mathcal{X}) = \vee_2 \psi(\mathcal{X}), \forall \mathcal{X} \subset \mathcal{L}_1$;
- ψ é uma *erosão* sse ψ comuta com o ínfimo, i.e., $\psi(\wedge_1 \mathcal{X}) = \wedge_2 \psi(\mathcal{X}), \forall \mathcal{X} \subset \mathcal{L}_1$;
- ψ é uma *anti-dilatação* sse $\psi(\vee_1 \mathcal{X}) = \wedge_2 \psi(\mathcal{X}), \forall \mathcal{X} \subset \mathcal{L}_1$;
- ψ é uma *anti-erosão* sse $\psi(\wedge_1 \mathcal{X}) = \vee_2 \psi(\mathcal{X}), \forall \mathcal{X} \subset \mathcal{L}_1$.

Denotamos a erosão, a dilatação, a anti-erosão e a anti-dilatação por $\varepsilon, \delta, \varepsilon^a$ e δ^a , respectivamente.

Um reticulado $\mathcal{L}$ possui uma *operação de negação* se existe um operador ν tal que

$$X \leq Y \Rightarrow \nu(Y) \leq \nu(X) \quad e \quad \nu(\nu(X)) = X. \quad (X, Y \in \mathcal{L}).$$

Em reticulados equipados com a operação de negação, a anti-erosão e a anti-dilatação podem ser representadas, respectivamente, por $\varepsilon^a = \nu\varepsilon$ e $\delta^a = \nu\delta$.

Sejam $\alpha \in \mathcal{L}_2^{\mathcal{L}_1}$ e $\beta \in \mathcal{L}_1^{\mathcal{L}_2}$. O par (α, β) é uma *conexão de Galois*^[i] entre $\mathcal{L}_1$ e $\mathcal{L}_2$ sse

$$X \leq_1 \beta(Y) \Leftrightarrow Y \leq_2 \alpha(X), \quad (\forall (X, Y) \in \mathcal{L}_1 \times \mathcal{L}_2).$$

Sejam $\alpha \in \mathcal{L}_2^{\tilde{\mathcal{L}}_1}$ e $\beta \in \tilde{\mathcal{L}}_1^{\mathcal{L}_2}$. Se (α, β) forma uma conexão de Galois então α é uma erosão de $\mathcal{L}_1$ em $\mathcal{L}_2$ e β é uma dilatação de $\mathcal{L}_2$ em $\mathcal{L}_1$ ([Ser88], Teorema 1.6).

Definimos a relação $(\alpha, \beta) \leq (\alpha', \beta')$, $(\alpha, \alpha' \in \mathcal{L}_2^{\mathcal{L}_1})$ e $(\beta, \beta' \in \mathcal{L}_1^{\mathcal{L}_2}) \Leftrightarrow$

$$\alpha'(X) \leq \alpha(X) \quad e \quad \beta(Y) \leq \beta'(Y), \quad (\forall (X, Y) \in \mathcal{L}_1 \times \mathcal{L}_2).$$

As funções em $\mathcal{L}_1^l$ serão denotadas pelas letras minúsculas a e b e chamadas *funções estruturantes*. Suponha que $\mathcal{L}_2$ possui uma família sup-geradora. Um operador $a \in \mathcal{L}_1^l$ será denominado *função estruturante com propriedade de dilatação* se existe uma dilatação $\delta \in \mathcal{L}_1^{\mathcal{L}_2}$ tal que $\forall y \in l, a(y) = \delta(y)$, veja Figura 3.1. A proposição abaixo é demonstrada em [Bar92], Prop. 7.2.

FIGURA 3.1. Função estruturante com propriedade de dilatação.

Proposição 3.4 (caracterização da dilatação e da erosão). *Existe uma bijeção entre o subconjunto de $\mathcal{L}_1^l$, cujos elementos são funções estruturantes que têm propriedades de dilatação, e o reticulado completo das Conexões de Galois entre $\tilde{\mathcal{L}}_1$ e $\mathcal{L}_2$. A bijeção*

[i] A conexão de Galois apresentada nesse texto pode ser encontrada em [Bir67].

considerada é dada por $a \mapsto (\varepsilon, \delta)$:

$$\begin{aligned}\varepsilon(X) &= \varepsilon_a(X) = \bigvee \{y \in l : a(y) \leq X\}, & (X \in \mathcal{L}_1), \\ \delta(Y) &= \delta_a(Y) = \bigvee \{a(y) : y \in l(Y)\}, & (Y \in \mathcal{L}_2)\end{aligned}$$

e por $(\varepsilon, \delta) \mapsto a$:

$$a(y) = a_{(\varepsilon, \delta)}(y) = \delta(y), \quad (y \in l).$$

Mais ainda, dados dois pares $(a, (\varepsilon, \delta))$ e $(a', (\varepsilon', \delta'))$ quaisquer, temos

$$a(y) \leq a'(y) \quad (y \in l) \Leftrightarrow (\varepsilon(X), \delta(Y)) \leq (\varepsilon'(X), \delta'(Y)), \quad ((X, Y) \in \mathcal{L}_1 \times \mathcal{L}_2).$$

◇

Em outras palavras, qualquer erosão ε ou dilatação δ , cujo contradomínio é um reticulado sup-gerador, pode ser caracterizada por uma função estruturante.

3.2. Aplicação em imagens binárias

Nesta seção apresentaremos uma aplicação da Proposição 3.4 no reticulado $\mathcal{P}(E)$.

Seja E um subconjunto de $\mathbb{Z}^2$. Sejam δ_b e ε_b operadores de $\mathcal{P}(E)$ em $\mathcal{P}(E)$ definidos por:

$$\begin{aligned}\delta_b(X) &= \bigcup \{b(x) : x \in X\} & (X \in \mathcal{P}(E)) \text{ e} \\ \varepsilon_b(X) &= \{x \in E : b(x) \subset X\} & (X \in \mathcal{P}(E))\end{aligned}$$

onde b é uma função estruturante definida sobre E . Seja $B + x = \{y + x, y \in B\}$ onde $+$ denota a soma vetorial em $\mathbb{Z}^2$. Algumas possíveis escolhas para funções estruturantes são:

I. *Invariante por Translação* (i.t.)

$$b(x) = B + x \quad (x \in E)$$

com $B \in \mathcal{P}(E)$ e $E = \mathbb{Z}^2$. O subconjunto B é chamado *elemento estruturante*;

II. *Quase Invariante por Translação* (q.i.t.)

$$b(x) = B + x \cap E \quad (x \in E)$$

com $B \in \mathcal{P}(E)$ e E é um subconjunto finito de $\mathbb{Z}^2$. Veja na Figura 3.2 um exemplo de dilatação usando uma função estruturante q.i.t.;

FIGURA 3.2. Dilatação binária com função estruturante q.i.t..

III. Baseada em grafos de vizinhança

$$b(x) = \{y \in E : d(y, x) \leq 1\} \quad (x \in E)$$

onde E é o conjunto de vértices de um grafo e $d(y, x)$ é a distância entre y e x neste grafo, definida no capítulo anterior. Veja na Figura 3.3 um exemplo de dilatação baseada em grafos de vizinhança^[ii].

FIGURA 3.3. Dilatação binária com função estruturante baseada em grafos.

Mais detalhes sobre funções estruturantes i.t. e q.i.t.^[iii] podem ser encontrados em [BB93], Capítulo 4.

^[ii] Esta figura está simplificada, a função estruturante é dada pelos vizinhos de cada vértice do grafo.

^[iii] As funções estruturantes q.i.t. também são chamadas de funções estruturantes condicionalmente i.t..

3.3. Aplicação em imagens em níveis de cinza

Nesta seção apresentaremos duas aplicações da Proposição 3.4 no reticulado K^E .

Antes de mostrarmos essas aplicações definiremos a soma e a subtração restritas a um conjunto finito de inteiros K .

Definição 3.5. *Seja K um intervalo $[0, k]$ de $\mathbb{Z}$ com $k > 0$.*

Para $v \in \mathbb{Z}$, definimos o operador $t \rightarrow t \dot{+} v$ em K como

$$\begin{cases} 0 \dot{+} v = 0 & \forall v \in \mathbb{Z}; \\ t \dot{+} v = 0 & \text{se } t > 0 \text{ e } t + v \leq 0; \\ t \dot{+} v = t + v & \text{se } t > 0 \text{ e } 0 \leq t + v \leq k; \\ t \dot{+} v = k & \text{se } t > 0 \text{ e } t + v > k. \end{cases}$$

Analogamente, para $v \in \mathbb{Z}$, definimos o operador $t \rightarrow t \dot{-} v$ em K por

$$\begin{cases} 0 \dot{-} v = 0 & \text{se } t < k \text{ e } t - v \leq 0; \\ t \dot{-} v = t - v & \text{se } t < k \text{ e } 0 \leq t - v \leq k; \\ t \dot{-} v = k & \text{se } t < k \text{ e } t - v > k; \\ t \dot{-} v = k & \forall v \in \mathbb{Z}. \end{cases}$$

◇

Primeiro exemplo

Neste exemplo apresentaremos a dilatação e a erosão em K^E como sendo uma aplicação da Proposição 3.4.

Sejam δ_b e ε_b operadores de K^E em K^E definidos por:

$$\begin{aligned} \delta_b(F) &= \vee \{b(f_{u,v}) : f_{u,v} \leq F, \forall (u,v) \in E \times K\} \quad (F \in K^E) \text{ e} \\ \varepsilon_b(F) &= \vee \{f_{u,v} : b(f_{u,v}) \leq F, \forall (u,v) \in E \times K\} \quad (F \in K^E), \end{aligned}$$

onde $f_{u,v}$ é definido no Exemplo 3.3 e b é uma função estruturante. Algumas possíveis escolhas para funções estruturantes são:

I. Invariante por Translação

$$b(f_{u,v}) = g(x - u) + v, \quad (x \in E, g \in \mathbb{Z}^B \text{ e } B \in \mathcal{P}(E))$$

onde $E = \mathbb{Z}^2$, veja Figura 3.4;

FIGURA 3.4. Dilatação de uma imagem em níveis de cinza, representada por função, por uma função estruturante i.t..

II. Quase Invariante por Translação

$$b(f_{u,v}) = g(x - u) \dot{+} v, \quad (x \in E, g \in K^B \text{ e } B \in \mathcal{P}(E))$$

onde $E = \mathbb{Z}^2$, veja Figura 3.5;

FIGURA 3.5. Dilatação de uma imagem em níveis de cinza, representada por função, por uma função estruturante q.i.t..

III. Baseada em grafos de vizinhança

$$b(f_{u,v}) = g_u \dot{+} v, \quad \text{onde}$$

$$g_u = \begin{cases} g(x) & \text{se } x \in \{y \in E : d(y, u) \leq 1\}, \\ 0 & \text{c.c.} \end{cases} \quad \text{com } g \in K^E,$$

onde E é o conjunto de vértices de um grafo e $d(y, x)$ é a distância entre y e x neste grafo. Veja na Figura 3.3 uma ilustração deste exemplo.

Segundo exemplo

Neste exemplo apresentaremos outras formas de caracterizar a dilatação e a erosão em K^E usando a Proposição 3.4, por funções estruturantes.

- I. A dilatação e a erosão por uma *função estruturante q.i.t.* podem ser descritas equivalentemente, respectivamente, pelas expressões: $\forall f \in K^E$,

$$\delta(f)(x) = \max\{f(y) \dot{+} g(x - y) : y \in B^t + x\} \quad (x \in E \text{ e } g \in K^B) \quad e$$

$$\varepsilon(f)(x) = \min\{f(y) \dot{-} g(y - x) : y \in B + x\} \quad (x \in E \text{ e } g \in K^B),$$

onde $B^t = \{-b : b \in B\}$ é o transposto de $B \in \mathcal{P}(E)$;

- II. A dilatação e a erosão por uma *função estruturante baseada em grafos de vizinhança* podem ser descritas equivalentemente, respectivamente, pelas expressões: $\forall f \in K^E$,

$$\delta(f)(x) = \max\{f(y) \dot{+} g(y) : d(y, x) \leq 1\} \quad (x \in E \text{ e } g \in K^E) \quad e$$

$$\varepsilon(f)(x) = \min\{f(y) \dot{-} g(y) : d(y, x) \leq 1\} \quad (x \in E \text{ e } g \in K^E),$$

onde E é o conjunto de vértices de um grafo e $d(y, x)$ é a distância entre y e x neste grafo.

CAPÍTULO 4

Extensão da “*MMach Toolbox*”

Usando os operadores de ínfimo, supremo e composição e os operadores elementares da Morfologia Matemática, definidos no capítulo anterior, podemos construir uma *linguagem formal* sobre um conjunto finito de reticulados completos ([BB92]). Esta linguagem é chamada *linguagem morfológica*. Uma implementação dessa linguagem é chamada *máquina morfológica*. Um exemplo de máquina morfológica é a “*MMach Toolbox*”, uma caixa de ferramentas da Morfologia Matemática implementada no sistema KHOROS.

Neste capítulo apresentaremos o sistema KHOROS e suas estruturas de dados para o armazenamento de imagens. Apresentaremos também a arquitetura da “*MMach Toolbox*” e, finalmente, mostraremos sua extensão para a manipulação de dados com suporte em grafos.

4.1. Plataforma de trabalho KHOROS

Nesta seção apresentaremos o sistema KHOROS e suas estruturas de dados para o armazenamento de imagens ([BBL], [RASW90], [BB92] e [BB94]).

KHOROS ([RASW90]) é um ambiente projetado para a pesquisa em Análise de Imagens. Ele foi criado no “*Department of Electrical and Computer Engineering*” da “*University of New Mexico*”, Albuquerque, USA, e tornou-se muito popular.

Uma das ferramentas desse sistema é o ambiente de programação visual *CANTATA*, uma linguagem orientada por fluxos de dados usando o conceito de grafos orientados. Cada

nó (*ícone* - pequena figura retangular) representa um procedimento (programa escrito na linguagem C) e cada arco representa um caminho por onde os dados fluem. Um nó é executado quando todos os dados de entrada estão disponíveis. O conjunto de nós interligados é chamado de *workspace*. Cada nó executado dispara uma linha de comando do sistema operacional *UNIX*. Veja na Figura 4.1 a janela do CANTATA mostrando um exemplo de workspace.

FIGURA 4.1. Janela do CANTATA mostrando um workspace.

O KHOROS possibilita dois tipos de armazenamentos de imagens, chamados *imagem implícita* e *imagem explícita* e definidos no Capítulo 1. Numa imagem implícita armazenamos todos os pixels da imagem num vetor com $l \times c$ posições, onde l é o número de linhas e c é o número de colunas da imagem (considerando uma imagem planar). Fazendo uma conta simples obtemos as coordenadas espaciais de um pixel armazenado numa imagem implícita. Na imagem explícita armazenamos pontos esparsos de uma imagem, com coordenadas espaciais e o valor do ponto. Estes tipos de armazenamentos de imagens são também chamados de *imagens multibandas*. Na imagem implícita, cada banda representa uma imagem em duas dimensões. Na imagem explícita, temos uma banda para cada coordenada espacial dos objetos e outra para armazenar os valores destes.

O KHOROS fornece um campo em sua estrutura de imagem para guardar o *tamanho do pixel*, onde o valor padrão é 1 para as projeções horizontal (largura) e vertical (altura) do pixel. O tamanho do pixel é dado pelo produto destas projeções.

As coordenadas horizontal e vertical numa imagem crescem da esquerda para a direita e de cima para baixo, respectivamente.

No KHOROS existe uma biblioteca com várias rotinas que estão divididas em duas classes principais: *vroutines* e *xvroutines*. Os programas do tipo *xvroutines* têm a característica de possuir uma interface gráfica própria (p.e. `put_mult`) enquanto os programas *vroutines* não apresentam esta característica (p.e. `input`).

Os programas criados pelos usuários (*vroutines* e *xvroutines*) podem ser organizados num subsistema independente chamado *toolbox*.

Um exemplo de *toolbox* é a “*MMach Toolbox*” ([BBL]).

4.2. Arquitetura da “*MMach Toolbox*”

Nesta seção apresentaremos a arquitetura da “*MMach Toolbox*” ([BBL] e [BB94]).

A “*MMach Toolbox*” usa operadores morfológicos caracterizados por funções estruturantes.

A arquitetura da “*MMach Toolbox*” está organizada em níveis de complexidade, como segue: operadores básicos, primeiro, segundo e terceiro nível de operadores. Os operadores do primeiro nível fazem uma chamada aos operadores do nível básico. Os operadores do segundo nível fazem um número finito de chamadas aos operadores do nível básico. Finalmente, os operadores do terceiro nível fazem um número não definido de chamadas aos operadores do nível básico.

Por exemplo, temos no nível básico o operador de dilatação, cuja fórmula está definida no segundo exemplo da Seção 3.3. O primeiro nível pode ser exemplificado pelo operador *abertura*, definido de K^E em K^E , dado por,

$$\gamma_b = \delta_b \varepsilon_b,$$

onde $K = [k_1, k_2]$, $k_1, k_2 \in \mathbb{Z}$, E é o domínio de uma imagem e b é uma função estruturante. No segundo nível temos, por exemplo, o operador *n-dilatação*, definido de K^E em K^E , dado pelas sucessivas $n - 1$ composições,

$$\delta_b^n = \delta_b \cdots \delta_b,$$

onde n é um número inteiro. Finalmente, exemplificamos um operador do terceiro nível com a *abertura por reconstrução* com marcador f , definidos de K^E em K^E , dado por,

$$\gamma_{b,f}(g) = \vee \{ \delta_{b,g}^n(f) : n = 1, \dots \},$$

onde $\delta_{b,g}^n(f)$ é a n -dilatação condicional definida em [BBL].

No Apêndice C mostramos todos os operadores existentes na “*MMach Toolbox*”.

Além da arquitetura hierárquica a “*MMach Toolbox*” opera orientada por dados, i.e., cada operador possui um único programa e conforme os dados de entrada executamos o operador correspondente. Exemplificando, se tivermos uma imagem implícita binária (cada pixel ocupa um bit) e um arquivo contendo uma função estruturante correspondente a esta imagem como entrada do programa que realiza a dilatação, será executada a dilatação binária entre imagens implícitas binárias. Veja nas Figuras 3.2 e 4.2 ilustrações desta dilatação. Se tivermos uma imagem implícita em níveis de cinza (cada pixel ocupa um byte), será executado o operador de dilatação sobre imagens em níveis de cinza. Assim, a “*MMach Toolbox*” usa o conceito de polimorfismo^[i].

FIGURA 4.2. Dilatação em imagem implícita.

4.3. A “*MMach Toolbox*” estendida

Estendemos as opções de funções estruturantes dos operadores disponíveis na “*MMach Toolbox*” pela introdução de funções estruturantes criadas a partir de grafos de vizinhança. No Apêndice A mostramos a estrutura de dados das funções estruturantes. No cabeçalho desta estrutura existe um rótulo que define qual função estruturante está sendo

^[i] Um programa é polimórfico quando executa uma determinada função para tipos de dados distintos.

utilizada, além de variáveis contendo suas dimensões. No caso de grafos, estas dimensões correspondem às coordenadas de um retângulo envolvendo os vértices (no caso planar).

Usaremos as definições de arquivo de medidas e imagens explícitas introduzidas no Capítulo 1. Os arquivos de medidas são armazenados na imagem multibanda, definida na primeira seção deste capítulo, onde cada banda armazena medidas sobre cada objeto.

A ordem dos objetos na imagem explícitas é definida da esquerda para a direita e de cima para baixo. A ordem das medidas nos arquivos de medidas e a ordem dos vértices do grafo na estrutura da função estruturante, no caso de grafos, segue a ordem dos objetos na imagem explícita.

4.4. Implementações básicas

Apresentaremos nesta seção programas básicos necessários para a extensão da “*MMach Toolbox*”.

No Apêndice D mostramos exemplos da documentação para esses programas.

Usaremos também as seguintes nomenclaturas nesta seção: *IR* para imagem rotulada, *IP* para imagens partição, *IC* para imagem em nível de cinza, *IB* para imagem binária (usaremos 0 e 1 como valores de uma *IB*), $\sum I$ para a soma dos valores dos pixels de uma imagem implícita *I* e *AMG*, *AMR* e *AMI* para arquivos de medidas globais, regionais e individuais, respectivamente.

4.4.1. Criação de arquivos de medidas

Com o objetivo de calcular medidas sobre imagens criamos três programas que têm como saída arquivos contendo estas medidas. Estes programas foram agrupados no *nível medidas* da “*MMach Toolbox*”, veja Tabela C.3. Criamos também neste nível dois estimadores de distribuições.

vshapglo - Medidas globais

Este programa calcula *medidas globais* de uma *IB* e gera o *AMG* contendo estas medidas definidas abaixo:

- a) **área:** $\sum IB$;
- b) **perímetro:** $\sum (IB - \varepsilon_B(IB))$, onde $\varepsilon_B(IB)$ é a erosão da IB pelo *elemento estruturante* 3×3 , ([BB94]);
- c) **número de componentes:** número de componentes conexas da IB .

vshapreg - Medidas regionais

Este programa calcula *medidas regionais* a partir de uma IC e uma IR (ou IP), gerando o AMR . Este arquivo contém para cada rótulo k da IR as seguintes medidas:

- a) **média:** soma dos níveis de cinza do rótulo k dividido pelo número de pixels deste rótulo;
- b) **mediana:** mediana do conjunto formado pelos níveis de cinza dos pixels pertencentes ao rótulo k ;
- c) **entropia:** $(-\sum_{i=0}^{nc} p_i \lg p_i)$ onde p_i é a probabilidade do nível de cinza $i \in \{0, \dots, nc\}$ estar no rótulo k e nc é nível de cinza de valor máximo;
- d) **valor máximo:** pixel de maior nível de cinza do rótulo k .

vshapind - Medidas individuais

Este programa calcula *medidas individuais* de uma IR (ou IP) onde para cada rótulo calculamos as seguintes medidas armazenadas no AMI :

- a) **área:** número de pixels do rótulo;
- b) **percentagem de área:** área dividido pelo total de pixels da IR ;
- c) **área real:** área vezes o tamanho do pixel;
- d) **perímetro:** usa o mesmo método descrito nas medidas globais;
- e) **baricentro horizontal:** média das coordenadas horizontais;
- f) **baricentro vertical:** média das coordenadas verticais;

- g) **rugosidade:** quadrado da diferença da área pelo perímetro;
- h) **pixel mais à direita:** maior coordenada horizontal do rótulo;
- i) **pixel mais à esquerda:** menor coordenada horizontal do rótulo;
- j) **pixel mais acima:** menor coordenada vertical do rótulo;
- k) **pixel mais abaixo:** maior coordenada vertical do rótulo;
- l) **projeção vertical:** diferença do pixel mais abaixo pelo pixel mais acima do rótulo;
- m) **projeção horizontal:** diferença do pixel mais à direita pelo pixel mais à esquerda do rótulo;

Se tivermos um grafo como entrada em `vshapind` podemos também calcular as seguintes medidas para cada vértice deste grafo:

- n) **número de vizinhos:** número de vizinhos do vértice;
- o) **distância:** assume 1 se o vértice possui pelo menos um vértice vizinho com distância euclidiana entre dois escalares ou assume 0, caso contrário.

vcovar - Covariância

Este programa calcula a *covariância* ([Ser82]) de uma imagem I . Esta medida é calculada por sucessivas translações de I em uma dada direção. A cada iteração calculamos a soma dos valores dos pixels da imagem transladada interseção com I , dividido pelo número total de pixels de I . O resultado é armazenado num vetor. O número de iterações ni e a direção d (ângulo em graus) são parâmetros de entrada. Para ser mais preciso, seja $T_d^i(I)$ a i -ésima translação de I na direção d onde $i \in \{0, 1, \dots, ni\}$. Estas translações iniciam com tamanho zero indo até o tamanho ni incrementado de uma unidade a cada iteração. A covariância, denotada por C , é definida pela seguinte fórmula:

$$C_d^i(I) = \frac{\sum (T_d^i(I) \cap I)}{N}$$

onde N é o número total de pixels de I .

vgran - Granulometria

Este programa calcula duas curvas *granulométricas* ([SS]) de uma imagem I . A primeira curva, denotada por $G1$, é definida da seguinte forma:

$$G1_b^i(I) = 1 - \frac{\sum \gamma_b^i(I)}{\sum I}$$

onde $\gamma_b^i(I)$ é a i -ésima abertura morfológica ([BBL]) da imagem I pela função estruturante b , $i \in \{0, 1, \dots, ni\}$, ni é um parâmetro de entrada e $\sum I$ não pode ter valor zero.

A segunda curva, denotada por $G2$, é definida como segue:

$$G2_b^i(I) = G1_b^i(I) - G1_b^{i-1}(I)$$

onde $i \in \{1, 2, \dots, ni\}$ e $G2_b^0(I) = 0$.

4.4.2. Criação de imagens explícitas

A criação de imagem explícita (IE) foi implementada nas seguintes rotinas, que teremos incluídas no *nível imagens explícitas* da “*MMach Toolbox*”, veja Tabela C.5. Os objetos da IE são armazenados na seguinte ordem: ordenamos as coordenadas espaciais verticais de cima para baixo. Se houver empate, ordenamos as coordenadas espaciais horizontais da esquerda para a direita.

vexplic1 - Imagem explícita 1

Este programa tem como entradas uma IB e uma IC . A saída é uma IE , definida por: a IB define as coordenadas e a IC define os valores dos objetos. Para cada pixel diferente de zero na IB extraímos o nível de cinza do pixel correspondente na IC .

vexplic2 e vexplic3 - Imagens explícitas 2 e 3

Temos como entradas do programa vexplic2 (vexplic3) um AMR (AMI) e uma IR (ou IP). Definimos uma IE por: o AMR (AMI) define os níveis de cinza e a IR define as coordenadas dos objetos. Para cada rótulo k da IR , encontramos as coordenadas dos objetos da IE , como segue:

- a) **baricentro:** $\sum(x/n, y/n)$, onde $\forall(x, y)$ pertence ao rótulo k e n é o número de pixels deste rótulo. Caso este valor esteja fora do rótulo k utilizamos o próximo método para determinar as coordenadas do objeto em IE referente ao rótulo k ;
- b) **mediana:** Seja vet^k um vetor contendo as coordenadas dos pixels de rótulo k em ordem da esquerda para a direita e de cima para baixo, seja n o número de pixels deste rótulo. Então a mediana é definida por $vet^k[\lfloor n/2 \rfloor]$.

vcplcd - Imagem explícita 4

Neste programa criamos uma IE a partir de duas $IE's$, uma fornece o nível de cinza e a outra as coordenadas dos objetos.

4.4.3. Criação de grafos de vizinhança

Nesta seção apresentaremos programas que criam grafos de vizinhança. Estes programas estão incluídos no *nível funções estruturantes* da “*MMach Toolbox*”, veja Tabela C.4.

vgstrfc1 - Adjacência

Este programa cria um grafo G a partir de uma IE e de uma IP , onde os vértices são definidos pela IE e as arestas pela IP . Uma aresta pertence ao grafo G se os seus extremos pertencem a partes adjacentes na IP .

vgstrfc2 - Distância máxima

Este programa cria um grafo G a partir de uma IE . Os vértices são definidos por IE e as arestas são aquelas que possuem distância Euclidiana não maior que um número inteiro (distância Euclidiana entre os objetos da IE).

vgstrfc3 - K vizinhos

Os vértices do grafo criado por este programa são definidos por uma IE e as arestas são as k mais próximas onde k é um inteiro positivo.

vgstrfc4 - Valores dos vértices dado por uma imagem explícita

Este programa apenas altera os níveis de cinza contido nos vértices de um grafo de vizinha. Temos como entrada uma *IE* e um grafo. *IE* define os valores dos vértices. O grafo de entrada define as coordenadas destes vértices e as arestas do grafo de saída.

4.4.4. Conversões de imagens explícitas e de grafos

Nesta seção iremos fazer o processo inverso feito pela criação de imagens explícitas. Isto é, iremos converter uma imagem explícitas em uma imagem implícito. Esta conversão está implementada no programa *vexpslade*.

Iremos também converter uma estrutura de grafos numa imagem implícita, usando o programa *vgslide*.

Estes dois programas estão descritos abaixo e incluídos no nível “*tools*” da “*MMach Toolbox*”, veja Tabela C.1.

vexpslide - Conversão de imagem explícita para implícita

Neste programa implementamos dois métodos para converter uma *IE*.

No primeiro método temos uma *IE* como entrada e como saída uma imagem implícita, com dimensões definidas como parâmetros de entrada. Esse programa irá acrescentar numa imagem implícita nula os objetos de *IE*.

No segundo método temos uma *IE* e uma *IR* como dados de entrada e como saída temos uma nova *IR* com os rótulos definidos pela *IR* de entrada, porém, com os níveis de cinza dos rótulos definidos pela *IE*. Cada rótulo da *IR* de entrada tem que corresponder a exatamente um objeto da *IE* com as suas coordenadas pertencentes a este rótulo.

vgslide - Conversão de grafo para imagem implícita

Para converter um grafo *G* para uma imagem implícita, gerado por exemplo, pelo programa *vgstrfc2*, criamos o programa *vgslide* tendo como entradas *G* e uma *IR* (*IR* é opcional), veja Figura 4.3. Se não tivermos uma *IR* como entrada será exibido uma imagem implícita ilustrando um grafo planar com espessura das arestas e tamanho dos vértices definidos como parâmetros de entrada. Caso contrário, será criado uma nova

IR com o valor dos rótulos definidos por G . Cada rótulo da IR de entrada tem que corresponder a exatamente um vértice de G .

FIGURA 4.3. Dilatação em imagem explícita (baseada em grafos).

4.4.5. Geração de pontos aleatórios

Nesta subseção e na seguinte apresentaremos programas auxiliares.

Apresentaremos a seguir os programas `vgshot1` e `vgshot2` com o objetivo de gerar pontos aleatórios (pixels com nível de cinza 255) em imagens implícitas. Estes programas estão no nível “*random*” da “*MMach Toolbox*”.

vgshot1 - Pontos aleatórios 1

Este programa particiona uma imagem implícita em retângulos de tamanhos iguais. Para cada retângulo são gerados pontos *pseudo-aleatórios* ([FM88] e [FB95]).

Um parâmetro de entrada desse programa é a porcentagem das dimensões dos retângulos com relação a imagem de entrada. Por exemplo, se este parâmetro valer 50% particionamos a imagem pela metade na largura e na altura obtendo quatro retângulos.

A intensidade de pontos k_i de cada partição $i \in \{1, \dots, n\}$, onde n é o número de partições, é obtido pela seguinte fórmula:

$$k_i = \min\{l \in \mathbb{Z}^+ : x_i \leq \sum_{j=1}^l I_j\}$$

onde $I_j = e^{-PA}(PA)^j/j!$, A é a área de um retângulo, P é a intensidade do processo (parâmetro de entrada) e x_i é uma variável pseudo-aleatória entre 0 e 1.

vgshot2 - Pontos aleatórios 2

Este programa gera pontos aleatórios numa imagem implícita I de entrada usando a seguinte fórmula: para cada pixel p de I temos,

$$p = \begin{cases} 255 & \text{se } x < P \\ 0 & \text{c.c.} \end{cases}$$

onde P é a intensidade de pontos aleatórios (parâmetro de entrada) e x é uma variável pseudo-aleatória entre 0 e 1.

4.4.6. Ferramentas auxiliares

Apresentaremos a seguir três programas incluídos no *nível “tools”*.

vminwin - Janela mínima

Este programa gera a menor imagem implícita contendo todos os pixels diferentes de zero de uma imagem implícita de entrada.

vwind - Borda ou fecho convexo

Se tivermos uma imagem explícita como entrada, este programa encontra o fecho convexo ([RS94]) dos objetos desta imagem. Os objetos pertencentes a este fecho recebem o valor 255 e os restantes recebem o valor zero. Agora, se tivermos uma imagem implícita como entrada este programa retorna uma imagem nula a menos da borda de espessura um que recebe o valor 255.

vrotate - Rotação

Este programa faz a rotação em graus de uma imagem implícita.

CAPÍTULO 5

Algoritmos para a extensão da “*MMach Toolbox*”

Neste capítulo apresentaremos algoritmos que constroem grafos de vizinhança. Apresentaremos também algoritmos para operadores morfológicos usando estes grafos ([Vin89], [BBL], [BS91], [Ser88], [SS95] e [BB92]).

Consideraremos neste capítulo imagens explícitas com n objetos (denotadas por *image_in*, *image_ou* e *image_mark*) e grafos de vizinhança com n vértices (denotado por *graph*), onde cada objeto corresponde a um vértice. Usaremos o conceito de conectividade em imagens explícitas significando conectividade em grafos em níveis de cinza, definida no Capítulo 2.

5.1. Grafos de vizinhança

Nesta seção apresentaremos algoritmos para encontrar grafos de vizinhança.

Os dois primeiros algoritmos que iremos apresentar fornecem arestas para um conjunto de vértices determinados por uma imagem explícita de entrada. As arestas são determinadas por uma imagem partição (com ou sem linha divisória, linha de espessura 1 dividindo partições adjacentes).

As coordenadas e os valores dos vértices são determinados pela imagem explícita de entrada.

5.1.1. Partição com linha divisória

Apresentaremos a seguir um algoritmo que encontra um conjunto de arestas usando uma imagem partição com linha divisória.

O algoritmo para encontrar estas arestas percorre a linha divisória e para cada pixels desta linha visita os vizinhos 4 ou 8 conectados (veja definição de 4 ou 8 conectado no Capítulo 2). Se estes vizinhos pertencerem a partições distintas, criamos arestas para os vértices correspondentes a estas partições.

Algoritmo 5.1 (partição com linha divisória).

entrada: *image_in* = imagem explícita
 image_rot = imagem partição com linha divisória
saída: *graph* = grafo de vizinhança

faça as dimensões de graph receberem as dimensões de image_rot;
para cada objeto *i* da *image_in*
 faça as coordenadas do vértice i de graph receberem as
 coordenadas do objeto i de image_in;
 graph[i] = image_in[i];
para cada pixel *j* da *image_rot*
 se *j* pertence a linha divisória **então**
 crie arestas para objetos, cujos níveis de cinza correspondem
 aos níveis de cinza distintos dos vizinhos (4 ou 8 conectados) de j;

◇

As notações *image_in[i]*, *image_ou[j]* e *graph[j]* referem-se ao objeto *i* da *image_in*, ao objeto *j* da *image_ou* e ao vértice *j* do grafo *graph*, respectivamente.

A complexidade deste algoritmo é $O(N)$, onde N é o número de pixels da *image_rot*.

Observe que este algoritmo só funciona se entre os pixels da imagem partição existir uma linha divisória.

5.1.2. Partição sem linha divisória

O segundo método para encontrar arestas tem uma imagem partição sem linha divisória como dado de entrada.

Para encontrar as arestas, iremos percorrer toda a imagem partição e para cada pixel visitar seus vizinhos 4 ou 8 conectados. Se estes vizinhos pertencerem a partições distintas, criamos arestas para os vértices correspondentes a estas partições.

Algoritmo 5.2 (partição sem linha divisória).

entrada: *image_in* = *imagem explícita*
 image_rot = *imagem partição sem linha divisória*
saída: *graph* = *grafo de vizinhança*

faça as dimensões de graph receberem as dimensões de image_rot;
para cada objeto *i* da *image_in*
 faça as coordenadas do vértice i de graph receberem as
 coordenadas do objeto i de image_in;
 graph[i] = image_in[i];
para cada pixel *j* da *image_rot*
 crie arestas para objetos, cujos níveis de cinza correspondem
 aos níveis de cinza distintos dos vizinhos (4 ou 8 conectados) de j;

◇

A complexidade deste algoritmo é igual a complexidade do algoritmo anterior.

5.1.3. Grafo distância máxima

O algoritmo que apresentaremos a seguir irá encontrar as arestas com distância $d(i, j)$ inferior a um escalar, onde i e j são os extremos destas arestas. Podemos usar distância *Euclidiana*, *City-Block* ou *Chessboard*. As coordenadas e os valores dos vértices são determinados por uma imagem explícita.

Algoritmo 5.3 (grafo distância máxima).

entrada: *image_in* = *imagem explícita*
 width = *largura do grafo*

$height$ = altura do grafo
 d = escalar
saída: $graph$ = grafo de vizinhança

faça as dimensões de graph receberem width e height;

para cada objeto i da $image_in$

faça as coordenadas do vértice i de graph receberem as coordenadas do objeto i de $image_in$;

$graph[i] = image_in[i]$;

para cada objeto j da $image_in$

se $i \neq j$ e $d(i, j) < d$ **então**

crie arestas para os vértices i e j ;

◇

A complexidade deste algoritmo é $O(n^2)$, onde n é o número de vértices do grafo.

5.1.4. Grafo dos k vizinhos mais próximos

Dado um conjunto de vértices. O algoritmo apresentado a seguir retorna os k vértices mais próximos, onde k é um número inteiro positivo.

Neste algoritmo iremos percorrer todos os vértices, definidos por uma imagem explícita, como segue: para cada vértice, criamos uma fila ordenada dos vértices restantes, com relação de distância *City-Block* (d_1). Em seguida encontramos os k vizinhos mais próximos nesta fila.

Algoritmo 5.4 (grafo dos k vizinhos mais próximos).

entrada: $image_in$ = imagem explícita
 $width$ = largura do grafo
 $height$ = altura do grafo
 k = inteiro positivo
saída: $graph$ = grafo de vizinhança

faça as dimensões de graph receberem width e height;

para cada objeto i da $image_in$

faça as coordenadas do vértice i de graph receberem as

coordenadas do objeto i de $image_in$;
 $graph[i] = image_in[i]$;
 crie uma fila ordenada dos vértices j ,
 com relação a $d_1(i, j)$, $\forall j \neq i \in image_in$;
 encontre os k vizinhos mais próximos nesta fila;

◇

A complexidade deste algoritmo é $O(n(O(n \lg n) + O(n)))$, ou seja, $O(n^2 \lg n)$.

5.2. Dilatação e Erosão

Os algoritmos apresentados a seguir, baseados em [Vin89] e [BBL], fazem a dilatação e a erosão de uma imagem explícita, $image_in$, por uma função estruturante baseada em grafos de vizinhança, $graph$, produzindo uma imagem explícita, $image_ou$. Estes algoritmos estão implementados nos programas *vdil* e *vero*, respectivamente. Os operadores de dilatação e de erosão usados estão definidos no segundo exemplo da Seção 3.3.

Nos algoritmos de dilatação e erosão, iremos percorrer todos os objetos de uma imagem explícita de entrada e para cada objeto visitar seus objetos vizinhos, determinados pela função estruturante, fazendo uma conta.

Usaremos a seguir os operadores $\dot{+}$ e $\dot{-}$ apresentados na Definição 3.5.

No algoritmo de dilatação, para cada objeto i da $image_in$ calculamos o máximo da soma $\dot{+}$ dos valores dos seus objetos vizinhos com os valores dos vértices correspondentes no grafo $graph$ atribuindo este cálculo ao objeto i da $image_ou$. Veja na Figura 5.1 uma ilustração do programa *vdil*. A estrutura de grafos usada neste capítulo será a $graph$ visualizada na Figura 5.2.

Algoritmo 5.5 (dilatação).

entrada: $image_in$ = imagem explícita
 $graph$ = estrutura de grafos
saída: $image_ou$ = imagem explícita

para cada objeto i da $image_in$
 $max_value = 0$;


```

para cada vizinho  $j$  de  $i$  em  $graph$ 
  se  $max\_value < image\_in[j] \dot{+} graph[j]$  então
     $max\_value = image\_in[j] \dot{+} graph[j]$ ;
 $image\_ou[i] = max\_value$ ;

```

◇

FIGURA 5.1. Dados de entrada e saída do programa *vdil* visualizados pelo programa *vexpslide*.

FIGURA 5.2. Estrutura de grafos armazenada em *graph* e visualizada pelo programa *vgslide*.

O algoritmo de erosão apresentado abaixo é semelhante ao algoritmo de dilatação, visto anteriormente, com a diferença de substituir o operador $\dot{+}$ pelo operador $\dot{-}$ e em vez de obter o máximo obtem-se o mínimo. Veja na Figura 5.3 uma ilustração do programa *vero*.

Algoritmo 5.6 (erosão).

entrada: *image_in* = imagem explícita
 graph = estrutura de grafos
saída: *image_ou* = imagem explícita

para cada objeto *i* da *image_in*
 min_value = ∞ ;
 para cada vizinho *j* de *i* em *graph*
 se *min_value* > *image_in*[*j*] - *graph*[*j*] então
 min_value = *image_in*[*j*] - *graph*[*j*];
image_ou[*i*] = *min_value*;

◇

image_in *image_ou*

FIGURA 5.3. Dados de entrada e saída do programa *vero* visualizados pelo programa *vexplide*.

Se *k* é o grau máximo do grafo *graph* então a complexidade dos algoritmos acima é $O(kn)$.

5.3. Função distância

O algoritmo que apresentaremos a seguir, implementado no programa *vdist*, retorna uma imagem explícita contendo distância em grafos. Cada objeto da imagem de saída recebe a distância em grafos com relação ao objeto de valor zero mais próximo da imagem de entrada. A vizinhança é determinada por um grafo.

A função distância ([Vin89]) de uma imagem explícita binária (os objetos desta imagem tem valores 0 ou 1) *image_in* com uma estrutura de vizinhança dada pelo grafo *graph* é uma imagem explícita em níveis de cinza *image_ou* definida por: para todo objeto *x* da *image_in*,

$$image_ou[x] = \min\{d(x, y) : image_in[y] = 0, \forall y \in E\}$$

onde *E* é o domínio da imagem e $d(x, y)$ é a distância em grafo entre *x* e *y*.

O primeiro passo desse algoritmo é rotular cada objeto da *image_in* com valor diferente de zero como *não_tratado*. Já os objetos com valor zero são rotulados como *tratado*. Esta rotulação é armazenada na *image_ou*. Os objetos com rótulo *tratado* são também armazenados numa fila. O segundo passo do algoritmo é retirar os objetos da fila, um a um, rotulando-os novamente com valor zero. Estes são os objetos com distância zero na *image_ou*. Inserimos novos objetos na fila com rótulo *não_tratado* e vizinhos dos objetos com rótulo *tratado* que acabaram de ser retirados da fila. Ao ser inserido na fila, estes objetos com rótulos *não_tratado* recebem rótulos *tratado*. No terceiro passo, retiramos da fila todos os objetos inseridos no passo anterior. Estes objetos são rotulados com valor um. Estes são os objetos com distância um na *image_ou*. Inseridos novos objetos na fila com rótulo *não_tratado* e vizinhos dos objetos que acabaram de ser retirados da fila. Novamente, ao ser inserido na fila o objeto com rótulo *não_tratado* recebe o rótulo *tratado*. Este processo continua até não haver mais objetos na fila. Veja na Figura 5.4 uma ilustração do programa *vdist*.

O algoritmo que iremos apresentar a seguir também é conhecido como *Dijkstra* ([Pre]).

Algoritmo 5.7 (função distância).

entrada: *image_in* = imagem explícita binária

graph = estrutura de grafos

saída: *image_ou* = imagem explícita em níveis de cinza

cont1 = *cont2* = *dist* = 0;

inicializa fila;

para cada objeto *i* da *image_in*

se *image_in*[*i*] = 0 então

coloque *i* na fila;

cont1++;

image_ou[*i*] = *tratado*;

senão

image_ou[*i*] = *não_tratado*;

```

enquanto fila não for vazia faça
  retire o primeiro objeto da fila e chame-o de i;
  se cont1 = 0 então
    {foram rotulados os objetos com distância dist}
    dist++;
    cont1 = cont2;
    cont2 = 0;
  image_ou[i] = dist;
  para cada vizinho j de i em graph
    se image_ou[j] = não_tratado então
      coloque j na fila;
      cont2++;
      image_ou[j] = tratado;
  cont1--;

```

◇

FIGURA 5.4. Dados de entrada e saída do programa *vdist* visualizados pelo programa *vexpslide*.

A complexidade desse algoritmo é $O(kn)$ onde k é o grau máximo do grafo *graph* e n é o número de vértices deste grafo. Este cálculo é obtido pela soma do tempo do primeiro laço **para** com o tempo do laço **enquanto**, i.e., $O(n) + O(nO(k))$, onde $O(k)$ é o tempo gasto pelo laço **para** dentro do laço **enquanto** quando é visitado os vizinhos de um objeto. Observe que em cada iteração do laço **enquanto** é retirado um objeto da fila e cada objeto é inserido na fila no máximo uma única vez, por esta razão aparece n no tempo gasto por este laço.

Esta complexidade pode ser melhorada para $O(m)$ onde m é o número de arestas.

Um outro algoritmo para a função distância pode ser obtido através da soma, objeto a objeto, de sucessivas erosões ([SS]). Porém este algoritmo não é tão eficiente quanto o apresentado acima, pois basta observar que cada erosão gasta $O(kn)$, como vimos na seção anterior.

5.4. Função distância geodésica

O problema da *função distância geodésica*, implementado no programa *vgeodist*, é semelhante ao problema da função distância apresentado na seção anterior. A diferença é que temos duas imagens explícitas binárias como entrada onde uma delas é uma imagem marcador (imagem binária) e em vez de começarmos a rotular os objetos diferentes de zero, vizinhos dos objetos iguais a zero, começamos a rotular os objetos diferentes de zero, vizinhos dos objetos diferentes de zero da imagem marcador.

O algoritmo para a função distância geodésica também é semelhante ao algoritmo da função distância apresentado anteriormente e por esta razão não apresentaremos aqui. Veja na Figura 5.5 uma ilustração da aplicação deste algoritmo.

FIGURA 5.5. Dados de entrada e saída do programa *vgeodist* visualizados pelos programa *vexpslide*.

5.5. Rotulações

O problema de *rotulação* ([Vin89]) tem como objetivo atribuir valores para as diferentes componentes conexas de uma imagem explícita onde a vizinhança dos objetos é dada por um grafo.

Implementamos no programa *vlabel* três tipos diferentes de algoritmos para a rotulação, com uma variável escolhendo qual algoritmo de rotulação deverá ser executado.

5.5.1. Rotulação binária

Neste primeiro algoritmo de rotulação, iremos atribuir, a cada componente conexa de uma imagem explícita de entrada, um nível de cinza distinto. A conexidade dos objetos é dada por um grafo cujos vértices correspondem aos objetos da imagem explícita.

Este algoritmo tem como entrada uma imagem explícita binária (imagem contendo objetos com valores 0 ou 1) *image_in* e uma estrutura de vizinhança definida por um grafo *graph*. Sejam $C_1, C_2, \dots, C_k$ as diferentes componentes conexas da *image_in*. A rotulação da *image_in* é uma imagem explícita em níveis de cinza *image_ou* definida por: para todo objeto x da *image_in*,

$$image_ou[x] = \begin{cases} 0 & \text{se } image_in[x] = 0 \\ k & \text{se } x \in C_k. \end{cases}$$

O algoritmo que apresentaremos a seguir inicialmente percorre cada objeto i da *image_in* fazendo *image_ou*[i] receber zero se *image_in*[i] for zero ou receber *não_tratado*, caso contrário. Em seguida o algoritmo percorre cada objeto i da *image_ou* e se este objeto foi rotulado como *não_tratado* o inserimos na fila rotulando com o valor um (primeira componente a ser rotulada). Agora, enquanto a fila não for vazia, retiramos o seu primeiro objeto, rotulamos com o valor um e visitamos seus vizinhos, caso estes tenham o rótulo *não_tratado* o inserimos na fila. Este processo se repete até todos os objetos desta componente contendo i receber o rótulo um. Em seguida tomamos o próximo objeto da *image_ou* com o rótulo *não_tratado* inserindo-o na fila e rotulando com o valor dois (segunda componente a ser rotulada). Enquanto a fila não for vazia, retiramos o seu primeiro objeto, rotulamos com o valor dois e visitamos seus vizinhos, caso estes tenham o rótulo *não_tratado* o inserimos na fila. Isto se repete até não haver mais objetos com o rótulo *não_tratado* na *image_ou*. Veja na Figura 5.6 uma ilustração da aplicação deste algoritmo.

Algoritmo 5.8 (rotulação binária).

entrada: *image_in* = imagem explícita binária
 graph = estrutura de grafos
saída: *image_ou* = imagem explícita em níveis de cinza

para cada objeto *i* da *image_in*
 se *image_in*[*i*] = 0 **então**
 image_ou[*i*] = 0;
 senão
 image_ou[*i*] = não_tratado;
 inicializa fila;
 rótulo = 1;
para cada objeto *i* da *image_ou*
 se *image_ou*[*i*] = não_tratado **então**
 coloque *i* na fila;
 image_ou[*i*] = rótulo;
 enquanto fila não for vazia **faça**
 { todos os objetos da componente contendo *i* receberão o valor rótulo }
 retire o primeiro objeto da fila e chame-o de *j*;
 image_ou[*j*] = rótulo;
 para cada vizinho *k* de *j* em *graph*
 se *image_ou*[*k*] = não_tratado **então**
 coloque *k* na fila;
 rótulo++;

◇

A complexidade desse algoritmo é $O(kn)$ onde k é o grau máximo de *graph*. Este tempo foi obtido pois temos $O(n)$ para o primeiro laço **para** e $O(nO(k))$ para o segundo. Este último tempo foi obtido fazendo uma análise amortizada ([CLR90]) onde observamos que para cada objeto da *image_ou* gastamos $O(k)$ para visitar seus vizinhos. Além disso, uma vez rotulado um objeto não será mais inserido na fila ou rotulado novamente.

Este algoritmo é uma versão do algoritmo de *Dijkstra* e gasta $O(m)$, onde m é o número de arestas do grafo.

FIGURA 5.6. Dados de entrada e saída do programa vlabel visualizados pelos programa vexpslide, usando a rotulação binária.

5.5.2. Rotulação em níveis de cinza

Esta rotulação difere da anterior na imagem de entrada, agora temos uma imagem explícita em níveis de cinza ao invés de binária. Iremos rotular cada componente conexa com um nível de cinza distinto.

O algoritmo para *rotulação em níveis de cinza* também é semelhante ao algoritmo apresentado na seção anterior, diferenciando apenas em dois pontos. No primeiro laço **para** rotulamos todos os objetos da *image_ou* como *não_tratado*. No último **se** acrescentamos a comparação $image_in[k] = image_in[i]$, com o objetivo de rotular todos os objetos com o mesmo nível de cinza em *image_in* numa mesma componente com um único rótulo.

Esse algoritmo tem a mesma complexidade do anterior, apesar de ser mais lento por um fator constante devido às duas comparações no lugar de uma no último **se**.

5.5.3. Rotulação genérica

Este problema de rotulação difere do anterior quando rotulamos duas componentes conexas distintas com níveis de cinza iguais com rótulos iguais.

O algoritmo que apresentaremos para resolver esse problema é semelhante ao algoritmo para a rotulação binária, visto anteriormente. A diferença é que ao rotular uma componente precisamos checar se já rotulamos uma outra componente com o mesmo nível de cinza na imagem de entrada *image_in*. Se isto ocorrer devemos rotular a componente

com o mesmo rótulo da componente já rotulada. Isto pode ser feito criando um vetor com tamanho igual ao nível de cinza máximo da *image_in* onde cada campo refere-se a um nível de cinza desta imagem. Inicialmente este vetor será inicializado com os campos valendo *não_tratado* e a medida que componentes conexas forem rotuladas os campos deste vetor deixam de ter o valor *não_tratado* para ter o valor do rótulo da componente na imagem de saída *image_ou*. Análogo à rotulação em níveis de cinza, rotulamos inicialmente todos os objetos na *image_ou* como *não_tratado* e no último se acrescentamos a comparação $image_in[k] = image_in[i]$. Veja abaixo um algoritmo para a rotulação genérica.

Algoritmo 5.9 (rotulação genérica).

entrada: *image_in* = imagem explícita binária

graph = estrutura de grafos

saída: *image_ou* = imagem explícita em níveis de cinza

maxval = rótulo = 0;

para cada objeto *i* da *image_in*

image_ou[*i*] = *não_tratado*;

se *image_in*[*i*] > *maxval* **então**

maxval = *image_in*[*i*];

crie um vetor *vet* com *maxval* posições inicializando-as com o valor *não_tratado*;

inicializa fila;

rótulo = 1;

para cada objeto *i* da *image_in*

se *image_ou*[*i*] = *não_tratado* **então**

coloque *i* na fila;

nível_cinza = *image_in*[*i*];

se *vet*[*nível_cinza*] ≠ *não_tratado* **então**

image_ou[*i*] = *vet*[*nível_cinza*];

senão

image_ou[*i*] = *vet*[*nível_cinza*] = *rótulo*;

enquanto fila não for vazia **faça**

{ todos os objetos da componente contendo *i* receberão o valor *rótulo*}

retire o primeiro objeto da fila e chame-o de *j*;

image_ou[*j*] = *rótulo*;

para cada vizinho *k* de *j* em *graph*

se *image_ou*[*k*] = *não_tratado* e *image_in*[*k*] = *nível_cinza* **então**

coloque *k* na fila;

rótulo++;

◇

A complexidade desse algoritmo é a mesma que as dos algoritmos de rotulação apresentados anteriormente.

5.6. Zonas de influência

O algoritmo de *zonas de influência* ([Vin89]) que iremos apresentar a seguir tem como entrada uma imagem explícita binária. Este algoritmo inicialmente rotula a imagem de entrada e em seguida atribui aos objetos, que ainda não foram rotulados, o valor do rótulo da componente mais próxima. Os objetos que têm distâncias iguais a duas ou mais componentes são rotulados com um nível de cinza distinto.

Seja *image_in* uma imagem explícita binária com uma estrutura de vizinhança dada por um grafo *graph*. Sejam $C_1, C_2, \dots, C_k$ as diferentes componentes conexas da *image_in*, a conexidade é determinada pelo grafo correspondente.

Para cada componente C_i , $i = 1, \dots, k$, desejamos encontrar as *zonas de influência* da *image_in*, $Z(C_i)$, i.e., os objetos da *image_in* que estão mais próximos de C_i do que de qualquer outra componente conexa da *image_in*. Para determinar as zonas de influência da *image_in* primeiro rotulamos *image_in* obtendo a imagem *rot(image_in)*. Em seguida para todo objeto x da *image_in* atribuímos a este objeto na *image_ou* o rótulo da componente conexa mais próxima na *rot(image_in)*. Se um objeto for igualmente distante de duas ou mais componentes conexas atribuímos o valor zero a este objeto na *image_ou*. Sendo mais preciso:

```

para cada objeto  $x$  da image_in
  se  $\exists i, \forall j \neq i, d(x, C_i) < d(x, C_j)$  então
    image_ou[ $x$ ] = rótulo de  $C_i$  na rot(image_in);
  senão
    image_ou[ $x$ ] = 0;

```

onde $d(x, C_i)$ é a distância em grafo de x à componente C_i e $i, j = 1, \dots, k$.

Veja abaixo um algoritmo em pseudo-código para encontrar as zonas de influência da *image_in* implementado no programa *vwatersh* da “*MMach Toolbox*”. Veja na Figura 5.7 uma ilustração deste programa.

Algoritmo 5.10 (zonas de influência).

entrada: *image_in* = imagem explícita binária
 graph = estrutura de grafos
saída: *image_ou* = imagem explícita em níveis de cinza

image_ou = rot(*image_in*); {*image_ou* recebe *image_in* rotulada}
 crie um vetor *vet_dist* de inteiros com *n* posições;
 {*n* é o número de objetos da *image_in*}
 inicializa fila;
cont1 = *cont2* = 0;
dist = 1;
para cada objeto *i* da *image_ou*
 se *image_ou*[*i*] ≠ 0 **então**
 vet_dist[*i*] = 0;
 senão
 vet_dist[*i*] = não_tratado;
 para cada vizinho *j* de *i* em *graph*
 se *image_ou*[*j*] ≠ 0 **então**
 coloque *i* na fila;
 vet_dist[*i*] = *dist*;
 cont1++;
 break;
enquanto fila não for vazia **faça**
 retire o primeiro objeto da fila e chame-o de *i*;
 se *cont1* = 0 **então**
 {todos os objetos com distância *dist* foram rotulados na *image_ou*}
 dist++;
 cont1 = *cont2*;
 cont2 = 0;
 image_ou[*i*] = não_tratado;
 para cada vizinho *j* de *i* em *graph*
 se *vet_dist*[*j*] = não_tratado **então**
 coloque *j* na fila;

```

    vet_dist[j] = dist + 1;
    cont2++;
senão
    se image_ou[j] != 0 e vet_dist[j] < dist + 1 então
        se image_ou[i] = não_tratado então
            image_ou[i] = image_ou[j];
        senão
            se image_ou[i] != 0 e image_ou[i] != image_ou[j] então
                image_ou[i] = 0;
cont1--;
se image_ou[i] = não_tratado então
    image_ou[i] = 0;

```

◇

FIGURA 5.7. Dados de entrada e saída do programa `vwatersh` visualizados pelo programa `vexpslide`.

A complexidade desse algoritmo é $O(kn)$ onde k é o grau máximo de *graph*. O primeiro passo desse algoritmo é fazer uma rotulação que gasta $O(kn)$, como vimos na Subseção 5.5.1. No Primeiro laço **para** percorremos todos os objetos da *image_ou* e para cada objeto visitado analisamos seus vizinhos, gastando assim $O(kn)$. Para cada iteração do laço **enquanto** rotulamos um objeto que ainda não foi rotulado e visitamos os seus vizinhos, gastando também $O(kn)$.

Esta complexidade pode ser melhorada para $O(m)$, onde m é o número de arestas do grafo.

5.7. Junção de regiões

O algoritmo que apresentaremos a seguir segmenta uma imagem explícita em níveis de cinza por um critério de proximidade, método “*region growing*” ([Cre93]).

O operador *junção de regiões* de uma imagem explícita em níveis de cinza *image_in* (cada região ou *zonas flats* ([Cre93] e [SS95]) é representada por um objeto desta imagem) necessita de um marcador dado por uma imagem explícita binária *image_mark* e de uma estrutura de vizinhança dada por um grafo *graph*. Este operador faz cada objeto da *image_in*, começando dos objetos diferentes de zero da *image_mark*, receber o nível de cinza de um dos seus vizinhos pelo critério de máxima proximidade dos níveis de cinza.

Um algoritmo para esse operador pode ser determinado pelos seguintes passos.

- (i) Todos os objetos diferentes de zero da *image_mark* são rotulados com o nível de cinza dos objetos correspondentes em *image_in* na imagem explícita de saída *image_ou*. Os objetos restantes são rotulados como *não_tratado*.
- (ii) Os objetos com o rótulo *não_tratado* e que são adjacentes a pelo menos um objeto que recebeu um nível de cinza no passo anterior são inseridos numa fila hierárquica. A prioridade de inserção é dada por uma medida de mínima dissimilaridade ou máxima proximidade calculada entre os valores dos objetos a serem inseridos e cada um de seus vizinhos que receberam um nível de cinza no passo anterior.
- (iii) **enquanto** fila não for vazia **faça**
 retire o primeiro objeto da fila e chame-o de *i*;
 se *image_ou*[*i*] = *não_tratado* **então**
 image_ou[*i*] recebe vizinho tratado com mínima dissimilaridade;
 todos os vizinhos de *i* que tem rótulo *não_tratado* são inseridos na fila pelo critério de mínima dissimilaridade;

Um algoritmo mais detalhado para resolver esse problema foi implementado no programa *vmosaic* e apresentado abaixo. Usaremos a notação $\|x\|$ como sendo o valor absoluto da variável inteira *x*. Veja na Figura 5.8 uma ilustração da aplicação do programa *vmosaic*.

Algoritmo 5.11 (junção de regiões).

entrada: *image_in* = imagem explícita em níveis de cinza
 image_mark = imagem explícita binária (marcador)

$graph$ = estrutura de grafos
saída: $image_ou$ = imagem explícita em níveis de cinza

```

 $image\_ou = rot(image\_mark);$  {  $image\_ou$  recebe  $image\_mark$  rotulada }
inicializa fila;
 $cont1 = cont2 = 0;$ 
 $dist = 1;$ 
para cada objeto  $i$  da  $image\_ou$ 
  se  $image\_ou[i] \neq 0$  então
     $image\_ou[i] = image\_in[i];$ 
  senão
     $image\_ou[i] = não\_tratado;$ 
para cada objeto  $i$  da  $image\_ou$ 
  se  $image\_ou[i] = não\_tratado$  então
     $var\_aux = \infty;$ 
    para cada vizinho  $j$  de  $i$  em  $graph$ 
      se  $image\_ou[j] \neq não\_tratado$  e  $\|image\_in[i] - image\_in[j]\| < var\_aux$  então
         $var\_aux = \|image\_in[i] - image\_in[j]\|$  ;
      associe  $var\_aux$  ao objeto  $i$ ;
      coloque  $i$  na fila de forma ordenada com relação aos valores associados;
enquanto fila não for vazia faça
  retire o primeiro objeto da fila e chame-o de  $i$ ;
  se  $image\_ou[i] = não\_tratado$  então
     $var\_aux = \infty;$ 
     $índice = -\infty;$ 
    para cada vizinho  $j$  de  $i$  em  $graph$ 
      se  $image\_ou[j] \neq não\_tratado$  e  $\|image\_in[i] - image\_in[j]\| < var\_aux$  então
         $var\_aux = \|image\_in[i] - image\_in[j]\|$  ;
         $índice = j;$ 
    se  $índice \geq 0$  então
       $image\_ou[i] = image\_ou[índice];$ 
    para cada vizinho  $j$  de  $i$  em  $graph$ 
      se  $image\_ou[j] = não\_tratado$  então
         $var\_aux = \infty;$ 
         $índice = -\infty;$ 
        para cada vizinho  $k$  de  $j$  em  $graph$ 
          se  $image\_ou[k] \neq não\_tratado$  e  $\|image\_in[k] - image\_in[j]\| < var\_aux$  então

```



```

var_aux = ||image_in[k] - image_in[j]|| ;
índice = k;
se índice >= 0 e image_ou[índice] = não_tratado então
coloque j na fila de forma ordenada com relação aos valores associados;

```

◇

FIGURA 5.8. Dados de entrada e saída do programa vmosaic visualizados pelos programa vexpslide.

A complexidade desse algoritmo é $O(k^2 n \lg n)$ onde k é o grau máximo de *graph*. O primeiro passo desse algoritmo é fazer uma rotulação que gasta $O(kn)$, como vimos anteriormente. No primeiro laço **para** varremos a imagem *image_ou* gastando $O(n)$. No segundo laço **para** percorremos a imagem *image_ou* e para cada objeto visitamos seus vizinhos e inserimos um objeto numa fila ordenada com no máximo $n - 1$ objetos usando busca binária, gastando $O(kn \lg n)$. No laço **enquanto** temos $O(n)$ iterações (um objeto é rotulado no máximo uma vez e em cada iteração retiramos um objeto da fila onde existem apenas objetos com rótulo *não_tratado*), para cada interação visitamos um objetos e seus vizinhos e para cada vizinho visitamos novamente seus vizinhos, gastando $O(k^2 n)$. Finalmente, gastamos $\lg n$ para inserir um objeto na fila ordenada. Assim, no laço **enquanto** temos $n(O(k) + O(kO(k)O(\lg n))) = O(k^2 n \lg n)$.

5.8. Zonas flats

O operador *zonas flats* ([SS95]) implementado no programa *vfats* é semelhante ao operador junção de regiões apresentado na Seção 5.7. Este operador tem como objetivo

unir regiões (definidas por máximos ou mínimos locais, veja [BBL]) por um critério de proximidade dos níveis de cinza.

Esse programa usa chamadas de programas da “*MMach Toolbox*”, funciona tanto para imagens implícitas quanto para imagens explícitas e o crescimento das regiões podem ser do maior para o menor nível de cinza, o contrário ou ambos. Este programa possui opcionalmente uma imagem binária como marcador e um parâmetro n que limita regiões através de n sucessivas erosões (as regiões são expandidas até certo ponto). O programa `vflats` possui um parâmetro N para o número de iterações, i.e., a cada iteração são encontrados os máximos locais ou os mínimos locais ([BBL]) e substituímos estes por um de seus vizinhos. Este processo se repete até completar as N iterações ou o método convergir. A convergência ocorre quando a imagem fica constante ou quando não se pode mais expandir as regiões devido aos marcadores (imagem marcador ou erosões sucessivas).

No caso de expandir regiões do menor para o maior nível de cinza encontramos os mínimos locais da imagem de entrada e substituímos estes pelo maior nível de cinza de um de seus vizinhos onde a vizinhança é dada pela função estruturante. No caso de expandir regiões do maior para o menor nível de cinza trocamos menor por maior e mínimos locais por máximos locais.

Mostramos na Figura 5.9 um workspace que visualiza as chamadas dos programas para o caso de expandir regiões do menor para o maior nível de cinza onde usamos 10 iterações no ícone `count_loop`, não usamos imagem marcador nem limites por erosões e usamos um elemento estruturante 3×3 cheio.

5.9. Filtro por área

Este operador tem como objetivo filtrar as componentes conexas de uma imagem implícita cuja a área de cada componente está entre dois limiares (números inteiros). Este operador foi implementado no programa `vfiltarea` usando chamadas de programas já definidos da “*MMach Toolbox*”.

Os passos para resolver este problema são: rotular a imagem de entrada; calcular a área de cada componente gerando um arquivo de medidas; criar uma imagem explícita correspondente à imagem de entrada e ao arquivo de medidas onde o valor dos objetos da imagem criada são as áreas das componentes conexas correspondentes; criar uma imagem implícita através da imagem explícita e da imagem rotulada criadas anteriormente; executar o programa `vthreshad` ([BBL]) tendo como parâmetros de entrada a imagem

FIGURA 5.9. Workspace visualizando os programas contidos em vflats.

criada no passo anterior e os limiares filtrando as componentes conexas que estão entre estes limiares. Estes passos podem ser visualizados através do workspace apresentado na Figura 5.10.

FIGURA 5.10. Workspace visualizando os programas contidos em vfilterarea.

CAPÍTULO 6

Aplicações da “*MMach Toolbox*” estendida

Neste capítulo apresentaremos várias aplicações do uso de operadores morfológicos baseados em grafos de vizinhança. Iremos tratar os seguintes problemas: cisalhamento de material metálico, segmentação da polpa do eucalipto, córnea humana, tecido muscular humano e segmentações em geral.

6.1. Material metálico

Considere uma imagem microscópica de uma liga metálica. Como encontrar uma possível linha de cisalhamento entre dois pontos pré-determinados?

Apresentaremos nesta seção uma solução para este problema usando operadores morfológicos baseados em grafos de vizinhança ([Vin90]).

Dada uma imagem sintética da superfície de um material metálico com buracos, Figura 6.1^[i], tentaremos localizar a linha de ruptura mais provável para um rompimento visto a maior concentração de buracos, ou seja, falta de material. Localizaremos a seqüência de pontos entre dois pontos escolhidos e o menor caminho entre eles é considerada a região mais provável para um cisalhamento.

^[i] As imagens das Figuras 6.1, 6.2, 6.3, 6.4 e 6.5 estão invertidas para melhor visualização.

FIGURA 6.1. Imagem sintética de um material metálico.

O primeiro passo para encontrar um cisalhamento na Figura 6.1 é achar o esqueleto ([BB94]) desta figura, mostrada na Figura 6.2.a. A seguir escolhemos dois pontos que serão os pontos utilizados para descobrir entre eles a linha mais provável para um cisalhamento. Uma forma de escolher estes pontos é rotular a Figura 6.2.a atribuindo para cada componente conexa um valor distinto. Para selecionar uma componente numa imagem rotulada utilizamos o programa *vtreshald* ([BBL]) que atribui o valor 255 à componente escolhida e zero às demais. Assim, geramos duas novas imagens com apenas um ponto em cada imagem valendo 255, os restantes valem zero. Na Figura 6.2.b mostramos os dois pontos escolhidos.

FIGURA 6.2. (a) Esqueleto e (b) Marcadores.

A partir da Figura 6.2.a geramos uma imagem explícita usando o programa *vexplic1*. Através do programa *vwatersh* encontramos as zonas de influência sobre o inverso da

Figura 6.2.a gerando a imagem I , veja na Figura 6.3.a o supremo de I com a Figura 6.2.a. Com a imagem explícita gerada anteriormente e a imagem I criamos um arquivo contendo uma estrutura de grafos através do programa `vgstrfc1`. Veja na Figura 6.3 a visualização deste grafo usando o programa `vgslide`. Geramos também duas imagens explícitas correspondentes a cada marcador.

FIGURA 6.3. (a) Supremo da imagem I com a Figura 6.2.a e (b) Grafo de vizinhança.

O próximo passo é encontrar a distância em grafos usando as imagens explícitas e o grafo criados anteriormente através do programa `vgeodist`. Este programa tem como entrada a imagem explícita correspondente ao esqueleto, Figura 6.2.a, uma imagem explícita servindo como marcador e a estrutura de grafos e gera uma imagem explícita onde cada objeto recebe um valor que corresponde à distância em grafos com relação à imagem marcador, ou seja, o objeto diferente de zero da imagem marcador recebe o valor zero, seus vizinhos recebem o valor um, os vizinhos dos que receberam o valor um, que não receberam o valor zero, recebem o valor dois e assim sucessivamente. Veja na Figura 6.4 as imagens mostrando a distância geodésica (Seção 5.4) com relação aos dois marcadores.

Usando o operador de adição `vaddm` (soma os valores dos objetos, em `[BBL]` este operador é definido sobre imagens implícitas) sobre as imagens geradas por `vgeodist`. Em seguida, escolhemos o menor valor da imagem gerada por `vaddm`, através do programa `vthreshld` (este programa seleciona objetos entre dois limiares, em `[BBL]` é definido sobre imagens implícitas). Para visualizarmos este resultado alteramos os valores dos vértices do grafo gerado por `vgstrfc1` pelos níveis de cinza dos objetos da imagem explícita criada no passo anterior usando o programa `vgstrfc4`. Para visualizar este grafo usamos o programa `vgslide`, veja Figura 6.5.

FIGURA 6.4. Distância em grafos com relação aos marcadores.

FIGURA 6.5. Imagem com uma possível linha de cisalhamento.

Um workspace para resolver o problema do cisalhamento em material metálico pode ser visto na Figura 6.6.

FIGURA 6.6. Workspace para encontrar cisalhamento em material metálico.

6.2. Eucalipto

Nesta seção apresentaremos uma aplicação da “*MMach Toolbox*” estendida numa imagem da polpa do eucalipto ([Jon96]), onde desejamos encontrar a área improdutiva da polpa do eucalipto, na industrialização da celulose.

Iremos segmentar a imagem microscópica do eucalipto, Figura 6.7, com o objetivo de encontrar a área de regiões com células adjacentes distantes.

FIGURA 6.7. Imagem da polpa do eucalipto.

O primeiro passo para essa segmentação é usar “*top hat*” ([Mey80]) sobre a Figura 6.7 seguido da abertura por reconstrução ([BB94]) gerando a Figura 6.8^[ii].

FIGURA 6.8. Segmentação usando “*top hat*” seguido da abertura por reconstrução.

^[ii] As imagens das Figuras 6.8, 6.9, 6.10 e 6.11 estão invertidas para melhor visualização.

A seguir usaremos o programa `wwatersh` sobre o inverso da Figura 6.8 gerando a imagem I . Na Figura 6.9 mostramos o supremo da Figura 6.8 com I .

FIGURA 6.9. Imagem mostrando o supremo da imagem I com a Figura 6.8.

Criamos uma imagem explícita usando o programa `vexplic3` tendo como entrada a rotulação da Figura 6.8 onde os níveis de cinza dos objetos da imagem explícita valem zero. A imagem rotulada será denotada por R . Construímos um grafo de vizinhança através do programa `vgstrfc1` tendo como entrada a imagem explícita criada anteriormente e I . Criamos um arquivo de medidas usando o programa `vshapind` tendo como entrada R e o grafo gerando anteriormente com o objetivo de encontrar medidas em grafos. Construímos uma nova imagem explícita usando o programa `vexplic3` tendo como entrada o arquivo de medidas criado no passo anterior e R , escolhendo a medida que atribui a cada objeto o valor da maior distância euclidiana dos objetos vizinhos, onde a estrutura de vizinhança foi determinada pelo grafo. Criamos uma nova imagem explícita através do programa `vthreshad` com o objetivo de filtrar os objetos com os maiores níveis de cinza (objetos considerando objetos vizinhos com distâncias maiores). Veja na Figura 6.10 a visualização desta imagem através do grafo onde os níveis de cinza dos vértices correspondem aos níveis de cinza dos objetos.

A seguir mostramos o resultado do fechamento (`[BBL]`) seguido da dilatação sobre a imagem explícita e o grafo criados anteriormente, veja Figura 6.11.

Finalmente, visualizamos o resultado anterior usando algumas facilidades da “*MMach Toolbox*”, veja Figura 6.12.

Um workspace para resolver o problema do eucalipto pode ser visto na Figura 6.13.

FIGURA 6.10. Grafo de vizinhança marcando vértices com arestas maiores.

FIGURA 6.11. Fechamento seguido de dilatação baseados em grafos.

FIGURA 6.12. Resultado da segmentação da imagem da polpa do eucalipto.

FIGURA 6.13. Workspace para segmentação da polpa do eucalipto.

6.3. Córnea

Neste experimento desejamos encontrar medidas que indicam a porcentagem de células mortas numa imagem microscópica da córnea humana. Este problema foi amplamente analisado em [Vin92] onde foi proposto um modelo matemático para encontrar o índice de mortalidade das células.

Para reproduzir os experimentos do artigo criamos duas imagens sintéticas, Figura 6.14, onde o nível de cinza de cada partição, contendo uma célula, corresponde a área desta partição. Nosso objetivo é verificar qual das duas imagens é mais doentia, i.e., tem a maior porcentagem de células mortas.

FIGURA 6.14. Duas imagens ilustrando partições contendo células da córnea. O nível de cinza corresponde a área da partição.

Para cada imagem I da Figura 6.14 criamos uma imagem explícita usando o programa `vexplic3` tendo como entrada a rotulação de I denotada por R . O nível de cinza dos objetos desta imagem explícita valem zero.

Construímos um grafo de vizinhança através do programa `vgstrfc1` tendo como entrada a imagem explícita criada anteriormente e R .

Criamos um arquivo de medidas usando o programa `vshapind` tendo como entrada R e o grafo correspondente com o objetivo de encontrar medidas em grafos.

Construímos uma nova imagem explícita usando o programa `vexplic3` tendo como entrada o arquivo de medidas e R , escolhendo a medida que atribui a cada objeto o número de objetos vizinhos. Veja na Figura 6.15 a visualização desta imagem explícita através do programa `vexpslide`.

Na Figura 6.16 mostramos passos intermediários do programa `vgran` que encontra a granulometria usando fechamentos de tamanho incremental tendo como entrada o inverso das imagens explícitas visualizadas na Figura 6.15 e os grafos correspondentes^[iii].

[iii] O programa `vgran` foi implementado usando aberturas, para obtermos fechamentos invertemos a imagem de entrada.

FIGURA 6.15. O nível de cinza das células corresponde ao número de vizinhos.

FIGURA 6.16. Granulometria usando fechamento incremental.

Os resultados de *vgran* podem ser vistos na Figura 6.17. Neles podemos analisar as imagens das córneas da Figura 6.14, onde observamos que o histograma à esquerda tem altura inferior ao histograma à direita, significando que a imagem correspondente à esquerda da Figura 6.14 tem o número de vizinhos das células mais regulares com relação à imagem da direita. Portanto, a imagem à esquerda é menos doentia que à imagem da direita ([**Vin92**]).

FIGURA 6.17. Histograma da granulometria.

Um workspace para encontrar o histograma de uma imagem da Figura 6.14 pode ser visto na Figura 6.18.

FIGURA 6.18. Workspace para análise de células da córnea.

6.4. Segmentações

Nesta seção mostraremos duas aplicações do programa *vmosaic* definido no capítulo anterior.

6.4.1. Segmentação do bloco

Apresentaremos a seguir a segmentação do bloco, Figura 6.19, usando grafos de vizinhança.

FIGURA 6.19. Imagem do bloco.

O primeiro passo é aplicar um filtro alternado seqüencial (podemos também usar um filtro alternado seqüencial conexos, veja [BB94]) sobre a Figura 6.19 com o objetivo de diminuir o número de zonas flats ([SS95] e [Cre93]), veja Figura 6.20.a. Rotulamos a imagem da Figura 6.20.a gerando a imagem da Figura 6.20.b.

Em seguida escolhemos algumas zonas flats que serão usadas como marcadores, veja Figura 6.21.

Criamos um arquivo de medidas usando o programa *vshapreg* referente às Figuras 6.19 e 6.20.b. Criamos uma imagem explícita com os níveis de cinza dos objetos iguais à média dos níveis de cinza da Figuras 6.19 correspondente às zonas flats da imagem da Figura 6.20.a usando o programa *vexplic2*. Criamos também um grafo de vizinhança através do programa *vgstrfc1* tendo como entrada a imagem explícita e a imagem rotulada mostrada na Figura 6.20.b. Através da imagem explícita criada, da imagem explícita correspondente aos marcadores e do grafo geramos a imagem segmentada do bloco, como mostra a Figura 6.22.

FIGURA 6.20. (a) Filtro alternado seqüencial e (b) Rotulação.

FIGURA 6.21. (a) Marcadores a mão e (b) Zonas flats marcadas.

FIGURA 6.22. Segmentação do bloco.

Um workspace para segmentações usando o programa vmosaic pode ser visto na Figura 6.23.

FIGURA 6.23. Workspace para segmentação usando o programa vmosaic.

6.4.2. Segmentação do fotógrafo

Apresentaremos a seguir a segmentação da imagem do fotógrafo, Figura 6.24.

A metodologia é análoga à apresentada na seção anterior.

FIGURA 6.24. Imagem do fotógrafo.

(a)

(b)

FIGURA 6.25. (a) Filtro alternado seqüencial e (b) Rotulação.

(a)

(b)

FIGURA 6.26. (a) Marcadores e (b) Zonas flats marcadas.

FIGURA 6.27. Segmentação do fotógrafo.

6.5. Músculo

A aplicação que apresentaremos a seguir tem como objetivo encontrar as células da imagem microscópica do músculo humano, Figura 6.28, com uma determinada característica. Iremos encontrar as células claras (com níveis de cinza próximo de 255) que têm células vizinhas escuras (com níveis de cinza próximo de 0).

Este problema exemplifica claramente a necessidade do uso da relação de vizinhança entre objetos (células), onde os operadores morfológicos baseados em grafos de vizinhança se encaixam perfeitamente.

FIGURA 6.28. Imagem microscópica do músculo humano.

O primeiro passo é segmentar a imagem da Figura 6.28 usando uma heurística apresentada em [Hir97]. O resultado desta segmentação pode ser vista na Figura 6.29.a. A seguir encontramos as zonas de influência desta segmentação usando o programa *vwatersh* ([BBL]). Seja I a imagem resultante deste programa. Veja na Figura 6.29.b o supremo da imagem segmentada com I .

(a)

(b)

FIGURA 6.29. (a) Segmentação da Figura 6.28 e (b) Supremo de I com a imagem à esquerda com níveis de cinza menores para melhor visualização.

Criamos um arquivo de medidas através do programa *vshapreg* tendo como entrada a imagem da Figura 6.28 e I . Com o arquivo de medidas e I criamos uma imagem explícita

através do programa *vexplic2* onde o nível de cinza dos objetos é o nível de cinza médio da célula correspondente da Figura 6.28. Para visualizarmos esta imagem explícita usamos o programa *vexpslide* criando a imagem da Figura 6.30.a.

Através da imagem explícita e de I construímos um grafo de vizinhança usando o programa *vgstrfc1*. Este grafo é visto na Figura 6.30.b usando o programa *vgslide*.

FIGURA 6.30. (a) Medidas regionais e (b) Grafo de vizinhança.

Executamos o programa *vtreshad* ([BBL]) sobre a imagem explícita visualizada na Figura 6.30.a, com o objetivo de segmentar as células mais claras (com níveis de cinza próximos a 255), gerando a imagem da Figura 6.31.a.

Encontramos o máximo local, programa definido em [BBL], da imagem explícita visualizada na Figura 6.30.a, com o objetivo de tentar selecionar novas células claras com células vizinhas escuras, criando a imagem da Figura 6.31.b. Este problema poderia ser resolvido usando apenas este passo, pois selecionamos células cujas células vizinhas tem níveis de cinza inferiores. Mas como vimos na Figura 6.31.b., este resultado não é satisfatório. Por exemplo, se tivermos uma célula com nível de cinza 10 com células vizinhas com níveis de cinza 9, todas estas células podem ser consideradas escuras. Os passos seguintes tentamos eliminar este caso.

Executamos novamente o programa *vtreshad* agora sobre a imagem da Figura 6.31.b, com o objetivo de limiarizar esta imagem, produzindo a imagem da Figura 6.32.a.

Fazemos a subtração da imagem da Figura 6.31.a com a imagem da Figura 6.32.a criando a imagem da Figura 6.32.b. Assim, selecionamos as células claras que não são máximos locais.

Dilatamos esta última imagem criando a imagem da Figura 6.33.a. Com isso, selecionamos as células claras que não são máximos locais e que possuem células vizinhas escuras (estas células vizinhas não são células claras rodeadas de células escuras, pois possuem pelo menos uma célula vizinha clara, determinada pelo primeiro limiar). Ou seja, esta

FIGURA 6.31. (a) “*Threshad*” da Figura 6.30.a e (b) Máximos locais da Figura 6.30.a.

FIGURA 6.32. (a) “*Threshad*” da Figura 6.31.b e (b) Subtração da Figura 6.31.a com a Figura 6.32.a.

nova imagem contém as células não procuradas. Invertendo esta imagem dilatada, Figura 6.33.b, encontramos outras possíveis células candidatas.

FIGURA 6.33. (a) Dilatação da Figura 6.32.b e (b) Negação da imagem à esquerda.

Fazendo o ínfimo das imagens das Figuras 6.31.a e 6.33.b criamos a imagem da Figura 6.34.a. Assim, refinamos ainda mais a solução, pois selecionamos as células clara que são máximos locais.

Observe que a partir da criação da imagem explícita, Figura 6.30.a, até este ponto, usamos apenas operadores sobre imagens explícitas e grafos.

Na Figura 6.34.b apresentamos uma outra forma de visualizar a imagem à esquerda.

FIGURA 6.34. (a) Ínfimo da Figura 6.31.a com a Figura 6.33.b e (b) Outra forma de ver a imagem à esquerda.

Aplicando essa heurística numa imagem maior obtemos a imagem da Figura 6.35.

Um workspace para resolver esse problema pode ser visto na Figura 6.36. Na Figura 6.37 mostramos o procedimento process do workspace anterior. O procedimento output visto no primeiro workspace contém recursos da “*MMach Toolbox*” para criar as imagens das Figuras 6.34.b. e 6.35.

A heurística que apresentamos para resolver o problema do músculo, pode ainda ser melhorada, pois existe casos onde as células são claras, máximos locais, mas tem alguma célula vizinha clara, com nível de cinza inferior à célula que é máximo local. Apesar disso, vimos claramente nesta seção, a manipulação de objetos com uma relação de vizinhança.

Um refinamento deste problema seria calcular uma nova medida nas *medidas individuais* sobre grafo, onde para cada vértice, calculamos a menor diferença em níveis de cinza dos objetos vizinhos. Em seguida, criamos uma nova imagem explícita contendo estas medidas. Usamos o programa *vthreshad* selecionando apenas objetos com nível de cinza maior que um escalar. Finalmente, fazemos o ínfimo deste resultado com a Figura 6.34.a.

FIGURA 6.35. Segmentação das células do músculo humano.

FIGURA 6.36. Workspace visualizando os passos para a segmentação das células do músculo.

FIGURA 6.37. Visualização do procedimento process do workspace anterior.

CAPÍTULO 7

Conclusão

Nexte texto foi mostrado uma extensão da “*MMach Toolbox*” ([BBL]) para o uso de operadores morfológicos baseados em grafos de vizinhança. Para que tivéssemos um trabalho completo, introduzimos alguns conceitos de Morfologia Matemática destacando a caracterização de operadores morfológicos por funções estruturantes. Exemplos de funções estruturantes são as baseadas em grafos de vizinhança. Com base nesta teoria foi possível criar a extensão da máquina morfológica “*MMach Toolbox*” baseada em grafos de vizinhança. Em seguida mostramos alguns recursos oferecidos pelo KHOROS para que esta extensão fosse possível, entre eles está o armazenamento explícito de imagens. Com estas bases bem definidas iniciamos as implementações da extensão da “*MMach Toolbox*”. Começamos criando arquivos de medidas sobre imagens e a partir destes arquivos criamos imagens explícitas e grafos de vizinhanças. Alteramos em seguida os operadores elementares *dilatação* e *erosão* para que suportassem também imagens explícitas e grafos. Nos preocupamos em deixar “*MMach Toolbox*” polimórfica no uso destes operadores, ou seja, executamos o operador correspondente aos dados de entrada dos programas (operador convencional ou baseado em grafos). Com estas alterações feitas, tivemos um número grande de operadores baseados em grafos na “*MMach Toolbox*”, pois a hierarquia foi preservada. Para que toda a “*MMach Toolbox*” fosse estendida para grafos, alteramos os operadores específicos do nível básico, inspirado no trabalho [Vin89]. Entre eles estão as *funções distâncias*, a *zona de influência* e as *rotulações*. Implementamos também *filtro por área*, *junção de regiões* e *zona flats*.

Para completar o texto apresentamos várias aplicações da extensão da “*MMach Toolbox*”. Descrevemos um método para encontrar uma possível linha de cisalhamento em

uma amostra de material metálico. Usamos os poros deste material para determinar os objetos e usamos uma triangulação para encontrar as vizinhanças destes objetos. Assim, foi possível resolver o problema usando operadores morfológicos sobre grafos. Uma outra aplicação deste operadores foi segmentar a área improdutiva na imagem microscópica da polpa do eucalipto, para a industrialização da celulose. Aqui, os objetos são as células do eucalipto e a vizinhança é determinada pela adjacência entre células. Para aplicar os operadores morfológico neste problema, usamos a relação de distância entre objetos. O problema da córnea é similar ao anterior, com a diferença de manipular número de vizinhos dos objetos para usar os operadores morfológicos. No problema do músculo, os objetos também são as células do músculo e a vizinhança é determinada por células adjacentes. Usando os operadores sobre grafos, tentamos encontrar células claras rodeadas de células escuras. As duas últimas aplicações apresentadas neste texto são problemas de segmentações em geral, onde segmentamos a imagem do bloco e a imagem do fotógrafo, usando a noção de máxima proximidades de regiões.

No Apêndice A apresentamos a estrutura de dados das funções estruturantes. No Apêndice B mostramos um estudo teórico sobre um algoritmo de triangulação de Delaunay que futuramente poderá ser incluído na *MMach Toolbox*. No Apêndice C apresentamos todos os operadores existentes na *MMach Toolbox*. Finalmente, no Apêndice D mostramos dois exemplos de documentos que podem auxiliar um pesquisador no uso da extensão da *MMach Toolbox*, dando exemplos práticos e informações técnicas sobre os programas implementados.

Nossas expectativas futuras para a continuação deste trabalho seria implementarmos esta extensão numa biblioteca independente, usar esta biblioteca na *MMach Toolbox* da nova versão do KHOROS (na Figura 7.1 mostramos a sua interface gráfica CANTATA), aprofundarmos os estudos em operadores morfológicos baseados em grafos, trabalharmos com três ou mais dimensões, etc.

FIGURA 7.1. Interface gráfica do KHOROS versão 2.1 - CANTATA

APÊNDICE A

Estrutura de dados das funções estruturantes

```
/* Struct. el. type */

#define MM_IMG      0
#define MM_MAT      1
#define MM_GRAPH    2


/* Node of a graph */

typedef struct {
    int n;                /* number of node neighbors */
    int val;              /* node value */
    int x;                /* node coordinates */
    int y;
    int *neig;            /* pointer to the first element
                           of an array of node indices */
} mm_node;
```

```
/* Graph based structuring function */
```

```
typedef struct {  
    int n;                /* graph number of nodes */  
    mm_node *nodes;       /* pointer to the first element of an  
                           array of nodes */  
} mm_graph;
```

```
/* Generic structuring element: */
```

```
typedef struct _gen_str {  
    int tag;  
    int m, n, p;          /* dimensions */  
    int x[3][3];          /* elementary matrix */  
    short *dat;           /* general matrix */  
    mm_graph g;           /* graph based str. el. */  
} mat_str;
```

APÊNDICE B

Algoritmo para a triangulação de Delaunay

Neste apêndice apresentaremos um algoritmo que resolve o problema da triangulação de Delaunay, definida na Seção 2.3. Detalharemos a corretude e o tempo de execução desse algoritmo ([Vin89], [RS94], [FC91], [GS85], [CE85], [Lee82] e [JRZ91]).

B.1. Algoritmo

Apresentaremos nesta seção um algoritmo linear que encontra a triangulação de Delaunay de um conjunto de pontos do plano P , denotado por $Del(P)$, a partir da triangulação de Delaunay de dois subconjuntos de P de igual tamanho, onde a união desses subconjuntos é P .

Sejam E e D subconjuntos de P de igual tamanho tal que $Del(E) \cap Del(D) = \emptyset$ e $E \cup D = P$. Por conveniência seja E os $\lfloor |P|/2 \rfloor$ pontos de P mais à esquerda e D os $\lceil |P|/2 \rceil$ pontos de P mais à direita com relação a abscissa (em caso de empate, conforme a ordenada).

Se e é uma aresta de $Del(E \cup D)$ com extremos em E e D então dizemos que $e \in E \cdot D$. Seja pp_i uma aresta de $Del(E)$. A primeira aresta à esquerda de pp_i , onde um dos extremos é p , é denotada por $esq(pp_i)$. Analogamente definimos $dir(pp_i)^{[i]}$.

^[i] Por [GS85] podemos encontrar $esq(pp_i)$ e $dir(pp_i)$ em tempo constante.

O algoritmo que vamos apresentar a seguir irá construir a triangulação de Delaunay do conjunto de pontos $E \cup D$ a partir da triangulação de Delaunay de E e D , supondo que não exista quatro ou mais pontos cocirculares em $E \cup D$ ([FC91]).

Algoritmo B.1 (triangulação de Delaunay).

entrada: $Del(E)$ e $Del(D)$

saída: $Del(E \cup D)$

1. $Del(E \cup D) = Del(E) \cup Del(D)$;
2. encontrar a tangente comum inferior pq e a tangente comum superior de E e D onde $p \in E$ e $q \in D$;
3. **enquanto** pq não é a tangente comum superior **faça**
 - (a) insira pq em $Del(E \cup D)$;
 - (b) {determinação do candidato pp_i em E }
$$pp_i = \text{esq}(pq);$$

$$pp_{i+1} = \text{esq}(pp_i);$$
se p_i está à esquerda de pq **então**
 - enquanto** p_{i+1} pertence ao círculo formado por p, q e p_i **faça**

$$\text{elimine a aresta } pp_i \text{ de } Del(E \cup D);$$

$$p_i = p_{i+1};$$

$$pp_{i+1} = \text{esq}(pp_i);$$
 - {senão, não há candidato em E }
 - (c) {determinação do candidato qq_i em D }
$$qq_i = \text{dir}(qp);$$

$$qq_{i+1} = \text{dir}(qq_i);$$
se q_i está à direita de qp **então**
 - enquanto** q_{i+1} pertence ao círculo formado por p, q e q_i **faça**

$$\text{elimine a aresta } qq_i \text{ de } Del(E \cup D);$$

$$q_i = q_{i+1};$$

$$qq_{i+1} = \text{dir}(qq_i);$$
 - {senão, não há candidato em D }
 - (d) {escolha da nova aresta pq }
se apenas pp_i é candidata **então**

$$pq = p_iq;$$
senão
se apenas qq_i é candidata **então**

$$pq = pq_i;$$

senão

se ambos são candidatos então

pq receber a aresta $(p_iq$ ou $pq_i)$ que
determina o maior ângulo entre $pq_i p$ ou pp_iq ;

senão

insira pq em $Del(E \cup D)$;

$\{pq$ é a tangente comum superior}

◇

A Figura B.1 ilustra um passo intermediário do algoritmo de combinar duas triangulações de Delaunay. As arestas tracejadas são eliminadas enquanto a aresta pq_i é acrescentada na nova triangulação.

FIGURA B.1. Ilustração de um passo do algoritmo da triangulação de Delaunay.

B.2. Corretude e desempenho

Apresentaremos a seguir alguns resultados teóricos para demonstrar a corretude do algoritmo da triangulação de Delaunay.

O próximo lema dará condição necessária e suficiente para encontrar a triangulação de Delaunay para um conjunto de pontos P .

Lema B.2. *A aresta pq é uma aresta do diagrama de Delaunay de P sse existe círculo que passa por p e q e não contém nem passa por nenhum outro ponto de P .*

◇

Prova. Seja pq uma aresta do diagrama de Delaunay de P então existe uma reta, uma semi-reta ou um segmento de reta não degenerado e e qualquer ponto c de e (excluindo seus extremos) dista igual de p e q , seja R essa distância. Qualquer outro ponto s de P , $s \neq \{p, q\}$, tem uma distância maior que R com relação a c . Logo, o círculo de centro c e raio R , passando por p e q , é livre de pontos em P .

Agora, se existe um círculo Γ de centro c e raio R , passando por p e q , que não contém nem passa por nenhum outro ponto de P , então $\{c\} = Z(p) \cap Z(q)$. Para provar que pq é uma aresta do diagrama de Delaunay de P , temos que mostrar que $|Z(p) \cap Z(q)| > 1$.

Vamos provar que se $\{c\} = Z(p) \cap Z(q)$ então existe $s \in P \setminus \{p, q\}$ tal que $c \in Z(s)$. Suponhamos por contradição que não exista tal s , ou seja, $\exists r > 0$ tal que o círculo Γ_r de centro c e raio r está contido em $Z(p) \cup Z(q)$, mas isso implica que o círculo Γ_r sem a borda, denotado por $\overset{\circ}{\Gamma}_r$, menos o ponto c , está contido em $\overset{\circ}{Z}(p) \cup \overset{\circ}{Z}(q)$, onde $\overset{\circ}{Z}(p)$ é formado por $Z(p)$ sem seus extremos, analogamente $\overset{\circ}{Z}(q)$ é formado por $Z(q)$ sem seus extremos. Isto é, $\overset{\circ}{\Gamma}_r \setminus \{c\} \subseteq \overset{\circ}{Z}(p) \cup \overset{\circ}{Z}(q)$, mas isso é um absurdo.

Suponhamos por contradição que $\{c\} = Z(p) \cap Z(q)$, então pelo parágrafo anterior, $\exists s \in P \setminus \{p, q\}$ tal que $c \in Z(s)$, logo $s \in \Gamma$, o que é uma contradição. Logo $|Z(p) \cap Z(q)| > 1$. ■

FIGURA B.2. Ilustração do Lema B.2.

Lema B.3. *Três pontos definem um triângulo em $\text{Del}(P)$ sse existe círculo que passa por eles e não contém nem passa por nenhum outro ponto de P . O centro do círculo é o vértice do diagrama de Voronoi correspondente a esse triângulo.* $\diamond$

FIGURA B.3. Ilustração do Lema B.3.

Teorema B.4. *Se pqr é um triângulo que ocorre numa triangulação de Delaunay do fecho convexo de P , então o ângulo $\widehat{prq}$ é máximo dentre todos os ângulos da forma $\widehat{psq}$, onde $s \in P$ e está no mesmo semiplano de r em relação a pq (Figura B.4).* $\diamond$

FIGURA B.4. Ilustração do Teorema B.4.

Teorema B.5. *Sejam $a, b, p, q \in P$ distintos dois a dois. A aresta $pq \in \text{Del}(P)$ sse b não pertence ao círculo formado pelos pontos apq , para quaisquer pares de pontos a e b de P , respectivamente à esquerda e à direita de pq (Figura B.5).* $\diamond$

FIGURA B.5. Ilustração do Teorema B.5.

Prova. Seja $\{C_t\}$ um conjunto de círculos passando por pqs , onde $s \in P \setminus \{p, q\}$. O parâmetro t é o valor na reta mediana a pq , com origem sobre o ponto médio de pq , e também t é o centro do círculo C_t .

Assim, $C_{-\infty}$ denota o semi-plano à esquerda de pq , C_0 denota o círculo com diâmetro pq e C_{∞} denota o semi-plano à direita de pq .

Note que os semi-discos desses círculos à esquerda de pq são estritamente decrescente (pela propriedade de inclusão) com t estritamente crescente, enquanto os semi-discos à direita de pq são estritamente crescente.

Logo, existe círculo livre de pontos passando por pq se e somente se todo ponto a de P à esquerda de pq , o círculo $C_t = apq$ tem um valor t menor que qualquer círculo qpb com b à direita de pq . ■

Lema B.6. *Seja P um conjunto de pontos do plano, $p \in P$ e l uma reta passando por p . Por conveniência assumimos que l é horizontal. Sejam $p_1, p_2, \dots, p_k$, $k \geq 1$, pontos de P que estão acima de l no sentido anti-horário, tal que $pp_i \in \text{Del}(P)$, $1 \leq i \leq k$. Se x é um ponto qualquer de l à direita de p , seja Γ_i , $1 \leq i \leq k$, o semi-círculo passando por pxp_i que está acima de l . Então a seqüência $\{\Gamma_i\}$ é unimodal, i.e., existe algum j , $1 \leq j \leq k$, tal que para $1 \leq i < j$ temos $\Gamma_i \supseteq \Gamma_{i+1}$, enquanto para $j \leq i < k$ temos $\Gamma_i \subseteq \Gamma_{i+1}$ (Figura B.6). ◇*

Prova. Primeiro mostraremos que se x_i denota a interseção mais à direita do círculo formado pelos pontos p, p_i e p_{i+1} , $i = 1, 2, \dots, k-1$, com l , então a seqüência de pontos $x_1, x_2, \dots, x_{k-1}$ move monotonicamente para esquerda. Considere p_i, p_{i+1} e p_{i+2} , três

FIGURA B.6. Ilustração do Lema B.6.

vizinhos consecutivos de p onde pp_i, pp_{i+1} e pp_{i+2} são arestas de Delaunay. Seja C_i o círculo formado por $pp_i p_{i+1}$ e C_{i+1} o círculo formado por $pp_{i+1} p_{i+2}$ (Figura B.7). O ponto p_{i+2} não pertence ao círculo C_i , e os pontos p_i e p_{i+2} são opostos com relação a pp_{i+1} . Assim C_i e C_{i+1} passam por p e p_{i+1} . Como p_{i+1} está à esquerda de p_i , segue que x_{i+1} também está à esquerda de x_i .

FIGURA B.7. Ilustração do Lema B.6.

Note que, se x_i está à direita de x , então $x \in C_i$ ou equivalentemente, $p_{i+1} \in \Gamma_i$, logo $\Gamma_i \supseteq \Gamma_{i+1}$ (Figura B.8.a). Após x_i mover para esquerda de x , $x \notin C_i$, o que implica $p_i \in \Gamma_{i+1}$, logo $\Gamma_i \subseteq \Gamma_{i+1}$ (Figura B.8.b). ■

Lema B.7. Se uma aresta de $\text{Del}(E \cup D)$ tem ambos os extremos em E (respectivamente D) então esta é uma aresta de $\text{Del}(E)$ (respectivamente $\text{Del}(D)$). ◇

FIGURA B.8. Ilustração do Lema B.6.

Lema B.8. *Quaisquer duas arestas E_D adjacentes na ordem do eixo y tem um vértice em comum e o terceiro lado do triângulo define uma aresta E_E ou D_D . $\diamond$*

Lema B.9. *Seja pq uma aresta de Delaunay onde $pq \in E_D$, e pp_i é a aresta válida no final da iteração do laço “enquanto”, linha 9 do algoritmo. Então o círculo formado por p, q e p_i é livre de pontos em E . $\diamond$*

Prova. Observando que p_i está à esquerda de pq . Considere o círculo C definido pelos pontos p, q (C é livre de pontos pois pq é aresta de Delaunay). Alteramos C de tal forma que passe sempre por p e q , e o centro de C mova sobre a mediatriz de pq , da direita para esquerda de pq , até que C toque num ponto $m \in P$ (Figura B.9.a). A aresta pm é de Delaunay em E , pois é livre de pontos em E . Desejamos mostrar que $p_i = m$. Seja C' o círculo formado por pqp_i . Se existe apenas uma aresta de Delaunay à esquerda de pq , então $p_{i+1} \notin C'$, logo $p_i = m$. Se existem várias aresta de Delaunay com p que estão à esquerda de pq , então pp_i itera até que $p_{i+1} \notin C'$ (Figura B.9.b). Os pontos $p_1, p_2, \dots, p_{i-1}$ formam arestas de Delaunay com $p \in E$ em $Del(E)$ e estão à esquerda de pq no sentido anti-horário e fora de C' , analogamente de C . Pela unimodalidade do Lema B.6, todos os pontos após p_{i+1} também estão fora do círculo C' , analogamente de C . Logo $p_i = m$. ■

Lema B.10. *As arestas $pp_1, pp_2, \dots, pp_{i-1}$ removidas no laço “enquanto”, linha 9 do algoritmo, não são arestas de Delaunay de $E \cup D$. $\diamond$*

Prova. Seja pq como definido no lema anterior. Quando o laço pp_i começa, sabemos que pp_i está à esquerda de pq . Então pp_{i+1} está à esquerda de pp_i ou à direita de pq (ou ambos). Se p_{i+1} pertence ao círculo formado por p, q e p_i então supondo que p_{i+1} está à direita de pq , usando o Teorema B.5 sobre pq obtemos que pq não é aresta de Delaunay, o

FIGURA B.9. Ilustração do Lema B.9.

que é uma contradição. Então pp_{i+1} está à esquerda de pq e à esquerda de pp_i . Aplicando novamente o Teorema B.5 sobre pp_i obtemos que pp_i não é aresta de Delaunay. Então pp_i é removido e a próxima aresta pp_{i+1} será avaliada. Por indução, o argumento se aplica a cada iteração. ■

Teorema B.11. *Após a execução do algoritmo, a subdivisão obtida é uma triangulação de Delaunay para $E \cup D$.* ◇

Prova. Primeiro vamos mostrar que a subdivisão é uma triangulação de $E \cup D$. A aresta pq começa como tangente comum inferior de E e D , e a cada iteração, a interseção de $Del(E)$ com $Del(D)$ separa uma linha de um ponto. O laço termina quando pq é a tangente comum superior de E e D . As arestas de outra parte de $Del(E)$ e $Del(D)$ nunca serão removidos e junto com o primeiro e o último pq constitui um fecho convexo de $E \cup D$. Os triângulos internos da subdivisão serão os triângulos originais de $Del(E)$ e $Del(D)$ ou novos triângulos delimitados por duas novas arestas E_D e uma aresta antiga E_E ou D_D .

Vamos mostrar que a triangulação é de Delaunay. A cada iteração determinamos pp_i tal que o círculo formado por p, q e p_i é livre de pontos em E (Lema B.9), simetricamente para qq_i . No final escolhemos entre p_iq e pqi o que é livre de pontos em $E \cup D$ (Figura B.10). Isto prova que o vencedor será a nova aresta de Delaunay. Concluimos que todas as arestas são de Delaunay, logo a subdivisão é uma triangulação de Delaunay. ■

Pelo Teorema 2.1, sabemos que o número de arestas e o número de triângulos numa triangulação com n vértices é linear em n . Então os passos (1) e (2) do algoritmo B1 gastam $O(n)$. As arestas inseridas em $Del(E \cup D)$, passo (3-a), vão da tangente comum

FIGURA B.10. Ilustração da demonstração do Teorema B.11.

inferior à tangente comum superior de $Del(E)$ e $Del(D)$, in order, gastando tempo constante. Fazendo uma análise amortizada ([CLR90]) dos laços em (3-b) e (3-c), verificamos que o número de repetições desses laços é proporcional ao número de arestas removidas, e cada aresta é removida no máximo uma vez. Então concluímos que o tempo gasto pelo passo (3) também é linear em n . Logo, o algoritmo para criar uma triangulação de Delaunay a partir de duas outras já existentes, gasta tempo $O(n)$.

Finalizando, dado um conjunto de pontos finitos do plano, onde não existem quatro ou mais pontos cocirculares, podemos achar a triangulação de Delaunay desses pontos, usando um algoritmo que usa o paradigma de divisão e conquista ([CLR90]). Inicialmente ordenamos os pontos com relação a abscissa e se houver empate, conforme a ordenada. Desta forma podemos dividir o conjunto de pontos em dois subconjuntos de igual tamanho em tempo constante. Encontramos a triangulação de Delaunay de cada subconjunto e fazemos a união desses, em tempo linear, como vimos anteriormente. Assim, o tempo de execução do problema de encontrar a triangulação de Delaunay pode ser dado pela recorrência $T(n) = O(n \lg n) + T'(n)$ onde $O(n \lg n)$ é o tempo gasto para ordenar os pontos e $T'(n)$ é o tempo gasto pelo algoritmo de divisão e conquista. A recorrência $T'(n)$ é dada por $T'(n) \leq 2T'(\lceil n/2 \rceil) + O(n)$ onde $T'(\lceil n/2 \rceil)$ é o tempo de execução para encontrar a triangulação de cada subconjunto e $O(n)$ é o tempo de execução para combinar as duas triangulações. A solução de $T'(n)$ é $O(n \lg n)$ podendo ser resolvido por indução ou por iteração ([CLR90]). Assim $T(n) = O(n \lg n)$ o que prova que o problema de encontrar a triangulação de Delaunay, que apresentamos, é resolvido em tempo $O(n \lg n)$.

APÊNDICE C

Operadores da “*MMach Toolbox*”

TABELA C.1. Nível “tools”.

Nome	Rotinas
Rot. Struct. Set	vrotstr
Slide graph	vgslide
Slide explicit	vexpslide
Equal	vequal
Window	vwind
Minimal Window	vminwin
Marder Window	vmarkwin
Rotate	vrotate
Less	vless
Cast1	vcast1

TABELA C.2. Nível “random”.

Nome	Rotinas
Corrupt image 1	vgshot1
Corrupt image 2	vgshot2

TABELA C.3. Nível medidas.

Nome	Rotinas
Global measures	vshapglo
Regional measures	vshapreg
Individual measures	vshapind
Covariance curve	vcovar
Granulometry curve	vgran

TABELA C.4. Nível funções estruturantes.

Nome	Rotinas
3x3 Struct. Set	vstrset
Disk Struct. Set	vgdisk
VIFF to Struct. Set	viff2str
Graph Struct. Set 1	vgstrfc1
Graph Struct. Set 2	vgstrfc2
Graph Struct. Set 3	vgstrfc3
Graph Struct. Set 4	vgstrfc4

TABELA C.5. Nível imagens explícitas.

Nome	Rotinas
Create Exp. Image 1	vexplic1
Create Exp. Image 2	vexplic2
Create Exp. Image 3	vexplic3
Copy Location Date	vcplocd

TABELA C.6. Operadores básicos de imagens.

Nome	Rotinas
Infimum	vinf
Supremum	vsup
Inversion	vinv
Subtraction	vsubm
Addition	vadd
Threshold	vtreshald
Toggle	vtoggle
Dilation	vdil
Erosion	vero
Distance	vdist
Geo. dist.	vgeodist
Labeling	vlabelexpan
Watershed	vwatersh
Exp. by dil	vdilexpan
Shrink. by dil	vdilshrin
Exp. by ero	veroexpan
Shrink. by ero	veroshrin
Filter area	vfiltarea
Flat zones in graph	vmosaic

TABELA C.7. Primeiro nível de operadores.

Nome	Rotinas
Anti-dilation	vadil
Anti-erosion	vaero
Gradient	vgradm
Cond. dilation	vcdil
Cond. erosion	vcero
Opening	vopen
Closing	vclose
Sup-generating	vsupgen
Inf-generating	vinfgn
Thinning	vthin
Thickening	vthick
Cond. thinning	vcthin
Cond. thickening	vcthick
Multiple inf	vminf
Multiple sup	vmsup
Symmetrical diff.	vsimdif

TABELA C.8. Segundo nível de operadores.

Nome	Rotinas
n-dilation	vndil
n-erosion	vnero
n-cond. dilation	vncdil
n-cond. erosion	vncero
n-opening	vnopen
n-closing	vnclose
n-open/close	vnocfilt
n-close/open	vncofilt
n-op/cl/op	vnocofilt
n-cl/op/cl	vncocvilt
n-thinning	vnthin
n-thickening	vnthick
n-cond. thin.	vcthin
n-cond. thick.	vcthick
n-canonical	vncanon
n-can. dual	vncanond
Erosion by a line	veroseg
Dilation by a line	vdilseg
Opening by a line	vopenseg
Closing by a line	vcloseseg
Dil. by a octagon	vdilocta

TABELA C.9. Terceiro nível de operadores.

Nome	Rotinas
Open. by rec.	vopenrec
Clos. by rec.	vclosrec
Center filter	vcenter
Skel. by thin.	vskelthin
Exoskel. by thick.	vskelthick
Cond. skel. thin.	vcskelthin
Cond. exosk. thick.	vcskelthick
Morph. skel.	vskel
Last erosion	vlastero
Cond. bisector	vbisset
Skel. by thin. mp	vskthinm
Close holes	vclohole
Edge off	vedgeoff
Homotopy	vhomot
Local min.	vlocmin
Local max.	vlocmax
Skiz	vskiz
Convex hull	vconvex
Flat zones	vmosaic

APÊNDICE D

Documentações para a “*MMach Toolbox*” estendida

D.1. Páginas na Internet

Existem documentos que ilustram os programas baseados em grafos da “*MMach Toolbox*” em páginas “*html*”, páginas que podem ser acessadas por computadores interligados pelo mundo através de softwares específicos.

Pode-se acessar essas páginas pelo endereço:

<http://www.ime.usp.br/~zampiol>

A Figura D.1 mostra um software que visualiza essas páginas.

FIGURA D.1. Página na Internet.

D.2. Ajuda "on line" do KHOROS

Outra documentação para os programas da "MMach Toolbox" estendida é a ajuda "on line" do KHOROS, fornecendo informações técnicas. Veja um exemplo através da Figura D.2.

FIGURA D.2. Ajuda "on line" do KHOROS.

Bibliografia

- [Bar92] J. Barrera. *Uma abordagem unificada para problemas de visão computacional: A morfologia matemática*. Tese de Doutorado em Engenharia Elétrica apresentada ao Departamento de Engenharia Eletrônica da EPUSP, São Paulo, 1992.
- [BB91] G.J.F. Banon and J. Barrera. Minimal representations for translation-invariant set mappings by mathematical morphology. *SIAM J. Appl. Math.* (51), 6:1782–1798, 1991.
- [BB92] J. Barrera and G.J.F. Banon. Expressiveness of the morphological language. *SPIE*, 1769:264–275, 1992.
- [BB93] G.J.F. Banon and J. Barrera. Decomposition of mappings between complete lattices by mathematical morphology; part 1: General lattices. *Signal Processing*, 30:299–327, 1993.
- [BB94] G.J.F. Banon and J. Barrera. *Bases da morfologia matemática para análise de imagens binárias*. IX Escola de Computação, Recife-PE, 1994.
- [BBL] J. Barrera, G.J.F. Banon, and R. A Lotufo. A mathematical morphology toolbox for khoros system.
- [Bir67] G. Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, Rhode Island, 1967.
- [BM76] J.A. Bondy and U.S.R. Murty. *Graph theory with applications*. The Macmillan Press LTD, London, 1976.
- [BS91] S. Beucher and J. Serra. Image segmentation by morphological methods. *Spie Educational Short Course*, 1991.
- [CE85] Bernard Chazelle and Herbert Edelsbrunner. An improved algorithm for constructing k^{th} - order Voronoi diagrams. *ACM*, 16:228–234, 1985.
- [CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., 1990.
- [Cre93] José Crespo. *Morphological Connected Filters and Intra-Region Smoothing for Image Segmentation*. A Thesis Presented to The Academic Faculty for Doctor of Philosophy in Eletrical Engineering, Georgia Institute of Technology, 1993.
- [FB95] P.A. Ferrari and J. Barrera. Comunicação verbal. *São Paulo - IME - SP*, 1995.

- [FC91] L.H. Figueiredo and P.C.P. Carvalho. *Introdução à Geometria Computacional*. Instituto de Matemática Pura e Aplicada, Rio de Janeiro-RJ, 1991.
- [FM88] P.A. Ferrari and E.S.R. Moura. A method to combine pseudo-random number generators using xor. *Relatório Técnico - São Paulo - IME - SP*, page 10, 1988.
- [GS85] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4 - No.2, Abril:74-123, 1985.
- [GW92] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 1992.
- [Hir97] R. Jr. Hirata. *Morphological Segmentation and Fast Algorithms*. Dissertação de Mestrado em Matemática Aplicada apresentada ao Departamento de Matemática Aplicada a Computação do IME/USP, São Paulo, 1997.
- [Jon96] Ronald Jones. A graph-based segmentation of eucalypt pulpwood images. *CSIRO Division of Mathematics and Statistics - North Ryde - Australia*, 1996.
- [JRZ91] M. Jünger, G. Reinelt, and D. Zepf. Computing correct Delaunay triangulations. *Computing*, 47:43-49, 1991.
- [Lee82] Der-Tsai Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE - Transactions on Computers*, C-31:478-487, 1982.
- [Mey80] F. Meyer. Quantitative analysis of the chromatin of lymphocytes: An essay on comparative structuralism. *Blood Cells* 6, pages 159-175, 1980.
- [Pre] F.P. and Preparata.
- [RASW90] J. Rasure, D. Argiro, T. Sauer, and C. Williams. Visual language and software development environment for image processing. *Imaging Systems and Technology*, 2:184-199, 1990.
- [Roc70] R.T. Rockafellar. *Convex Analysis*. Princeton, 1970.
- [RS94] P.J. Rezende and J. Stolfi. *Fundamentos de Geometria Computacional*. IX Escola de Computação, Recife-PE, 1994.
- [Ser82] J. Serra. *Image analysis and mathematical morphology*, volume 1. Academic Press, London, 1982.
- [Ser88] J. Serra. *Image analysis and mathematical morphology - Theoretical advances*, volume 2. Academic Press, London, 1988.
- [SS] P. Salembier and J. Serra. Morfologia matemática. *Dept. de Teoria de la Señal y Comunicaciones, Universid Politécnica de Cataluña. Centre de Morphologie Mathématique, Ecole des Mines de Paris*.
- [SS95] P. Salembier and J. Serra. Flat zones filtering, connected operators, and filters by reconstruction. *IEEE-Transactions on image processing*, 4 - No 8:1153-1160, August 1995.
- [Vin89] L. Vincent. Graphs and mathematical morphology. *Signal Processing*, 16:365-388, 1989.
- [Vin90] L. Vincent. *Algorithmes Morphologiques A Base de Files D'Attente et de Lacets. Extension Aux Graphes*. These présentée à l'Ecole Nationale Supérieure des Mines de Paris par obtenir le titre de Docteur en Morphologie Mathématique, 1990.
- [Vin92] L. Vincent. Morphological image processing and network analysis of cornea endothelial cell images. *SPIE*, 1769:212-226, 1992.

Índice Remissivo

- ícone, 36
- ínfimo, 26
- "MMach Toolbox"*, 1, 35

- anti-dilatação, 27
- anti-erosão, 27
- aresta, 11
 - laço, 12
- arestas
 - adjacentes, 12
 - paralelas ou múltiplas, 12
- arquivo
 - medidas, de, 7, 39

- CANTATA, 35
- caracterização da dilatação e da erosão, 28
- conectividade
 - 4, 12
 - 8, 12
- conexão de Galois, 28
- conjunto parcialmente ordenado, 26
- coordenadas cartesianas, 14
- covariância, 41

- diagrama de Delaunay, 17
- diagrama de Voronoi, 17
- dilatação, 27, 51
- disco digital, 16
- distância, 15
 - Chessboard, 15
 - City-Block, 15
 - Euclidiana, 15

- elemento estruturante, 6, 29
- erosão, 27, 52

- família sup-geradora, 26
- fecho convexo, 16
- filtro por área, 67
- função distância, 54
- função distância geodésica, 56
- função estruturante, 6, 28
 - baseada em grafos de vizinhança, 29, 32, 33
 - i.t., 29, 31
 - propriedade de dilatação, com, 28
 - q.i.t., 29, 32, 33

- Geometria Computacional, 14
- grade
 - hexagonal, 12
 - quadrada com vizinhança-4, 12
 - quadrada com vizinhança-8, 12
- grafo, 11
 - k -regular, 12
 - árvore geradora mínima, 19
 - binário, 13
 - caminho em, 12
 - conexo, 12
 - desconexo, 12
 - distância em, 12
 - distância máxima, de, 20, 49
 - finito, 12
 - Gabriel, de, 18
 - k vizinhos mais próximos, dos, 21, 50
 - níveis de cinza, em, 13
 - passeio em, 12
 - comprimento do, 12
 - quase regular, 12
 - referência, de, 13
 - regular, 12
 - similar, 13

- simples, 12
- subgrafo do, 12
- vizinhança relativa, de, 19
- vizinhança, de, 7, 13
- vizinhos do baricentro, dos, 21
- grafo distância máxima, 49
- grafo dos k vizinhos mais próximos, 50
- granulometria, 42
- imagem
 - binária, 27
 - explícita, 5, 6, 36
 - implícita, 6, 36
 - multibanda, 36
 - níveis de cinza, em, 26
 - partição, 7, 22
 - rotulada, 7, 22
- imagens
 - partição, 7
- junção de regiões, 64
- KHOROS, 35
- limitante
 - inferior, 26
 - superior, 26
- linguagem formal, 6, 35
- linguagem morfológica, 35
- máquina morfológica, 6, 35
- medidas
 - globais, 39
 - individuais, 7, 40
 - regionais, 7, 40
- Morfologia Matemática, 5, 25
- nível
 - “random”, 45
 - “tools”, 44, 46
 - função estruturante, 43
 - imagens explícitas, 42
 - medidas, 39
- objetos, 6
- operação de negação, 27
- operador, 26
 - abertura, 37
 - abertura por reconstrução, 38
 - elementar, 5, 25
 - morfológico, 5
 - n-dilatação, 37
- páginas “html”, 117
- partição com linha divisória, 48
- partição sem linha divisória, 49
- pixels, 2
- plano
 - cartesiano, 14
 - face do, 16
 - partição do, 16
- ponto, 14
 - à direita de uma reta, 15
 - à esquerda de uma reta, 15
 - exterior ao círculo, 15
 - interior ao círculo, 15
 - médio, 14
- pontos
 - colineares, 15
- poset, 26
 - dual, 26
- programa
 - vcovar, 41
 - vcplod, 43
 - vdil, 51
 - vdist, 53
 - vero, 51
 - vexplic1, 42
 - vexplic2, 42
 - vexplic3, 42
 - vexpslide, 44
 - vfiltarea, 67
 - vflats, 66
 - vgeodist, 56
 - vgran, 42
 - vgshot1, 45
 - vgshot2, 46
 - vgslide, 44
 - vgstrfc1, 43

- vgstrfc2, 43
- vgstrfc3, 43
- vgstrfc4, 44
- viwind, 46
- vlabel, 57
- vminwin, 46
- vmosaic, 64
- vrotate, 46
- vshapglo, 39
- vshapind, 40
- vshapreg, 40
- vwatersh, 62
- pseudo-aleatórios, 45
- região de Voronoi, 17
- reta, 14
 - oposta, 14
 - passando por dois pontos, 14
 - segmento de, 15
 - semi-, 15
- reticulado completo, 26
- rotulação, 57
- rotulação binária, 57
- rotulação em níveis de cinza, 59
- rotulação genérica, 60
- supremo, 26
- tamanho do pixel, 37
- Teoria dos Grafos, 11
- toolbox, 37
- triângulo, 16
 - degenerado, 16
 - fronteira do, 16
- triangulação de Delaunay, 18, 102
- UNIX, 36
- vértice, 11
 - branco, 13
 - conectado, 13
 - conectado a um conjunto de vértices, 13
 - grau ou número de vizinhos do, 12
 - incidente, 12
 - preto, 13
- vértices
 - adjacentes, 12
 - conjunto de
 - componente conexa do, 13
 - conectado a um conjunto de vértices, 13
 - conexo, 13
- vrotines, 37
- workspace, 36
- xvrotines, 37
- zonas de influência, 61, 62
- zonas flats, 64, 66