Segmentação de Imagens por Morfologia Matemática

Roberto Hirata Jr.

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, Março de 1997 -

Segmentação de Imagens
por
Morfologia Matemática

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e apresentada por Roberto Hirata Jr. e aprovada
pela Comissão Julgadora.

São Paulo, 7 de março de 1997

Banca Examinadora :

- Prof. Dr. Junior Barrera (orientador) - IME-USP
- Prof. Dr. Nelson D. d´Avila Mascarenhas - UFSCar
- Prof. Dr. Antônio Elias Fabris - IME-USP

aos meus pais e à Nina

Agradecimentos

Gostaria de agradecer a todas as pessoas que tive a sorte de conhecer durante estes anos de mestrado e a todos os meus amigos, pois este trabalho teria sido bem mais difícil sem a ajuda e apoio deles.

Em especial, quero dizer que:

tenho muito que agradecer ao amigo e orientador Junior Barrera por tudo que dele aprendi e com ele realizei durante o meu mestrado.

quero agradecer com muito carinho ao amigo e professor Valdemar W. Setzer que tem-me orientado e co-orientado desde 1986.

tenho muito a agradecer à minha amiga Nina S. Tomita pela companhia, confiança e apoio durante todos estes anos e, também, por ler e ajudar-me na preparação deste trabalho.

agradeço aos professores Nelson D. d'Avila Mascarenhas e Antônio Elias Fabris por terem participado e colaborado com diversas sugestões desde o meu exame de qualificação.

agradeço também aos professores Routo Terada, Flávio Soares Corrêa da Silva e Roberto de Alencar Lotufo pelos conselhos e pela oportunidade de trabalhar em equipe com eles.

agradeço aos amigos Francisco de Assis Zampirolli, João Kogler pelas diversas discussões na área e fora dela também.

agradeço aos amigos Adriano N. Rodrigues e Ricardo Ueda por todas as sugestões e ajuda na administração da rede do laboratório de Visão Computacional IME-OLIVETTI.

agradeço os funcionários e funcionárias da biblioteca, da seção de cópias, da secretaria do departamento de computação e da secretaria de pós-graduação por todo o trabalho que dei a eles.

agradeço aos meus tios Kazuo e Saiti Hirata que, desde pequeno, incentivaram-me nos estudos.

agradeço ao CNPq e à Olivetti do Brasil pelo apoio financeiro recebido durante a elaboração deste trabalho.

agradeço muito a minha família por todo o apoio e incentivo durante toda a minha vida.

À todos voces o meu muito obrigado !

Resumo

Este trabalho mostra o que é segmentação e como segmentar um imagem usando Morfologia Matemática (*MM*). Ele começa introduzindo o assunto segmentação pela abordagem clássica e depois passa a tratar a segmentação sob a abordagem da (*MM*). Uma pequena introdução à (*MM*) é dada e depois são abordados duas técnicas de segmentação morfológica, uma heurística e uma não heurística. A equivalência entre as abordagens clássicas e as morfológicas são mostradas e depois são apresentados diversos exemplos práticos de segmentação sob a abordagem morfológica. O trabalho também mostra alguns algoritmos importantes para a segmentação de imagens e como usamos as idéias por detrás deles para implementar eficientemente os operadores e operações elementares da (*MM*). Essa abordagem permite uma flexibilidade maior para implementar operadores mais complexos eficientemente. Finalmente são apresentados alguns resultados experimentais que confirmam estas idéias.

Abstract

This work shows what segmentation is and how to segment an image using Mathematical Morphology (*MM*). It begins by introducing the subject segmentation by the classical approach and then by the morphological one. A short introduction to *MM* is given and then two morphological approaches are given for segmentation: an heuristic and a non-heuristic one. The equivalence between the classical and the morphological approaches is shown and several illustrative examples of the morphological approach are given. The work also shows some important fast algorithms which simulate some of the operators used in the segmentations and how to use them to segment images. Finally it shows how to use the ideas behind the fast algorithms to implement efficiently the elementary operations and operators of *MM* so other complex operators can be implemented more efficiently than before. Some experimental results are shown to verify this assumption.

Índice

Lista de Figuras

Capítulo 1

Introdução

O conceito intuitivo que temos de uma imagem é que ela é uma representação visual estática do instantâneo de uma cena. Este conceito, apesar de correto, não é completo pois há outras formas de representação além daquela ligada à visão.

No nosso trabalho, estamos interessados na representação numérica de uma cena, mais especificamente, numa representação onde cada ponto ou, pelo menos, cada amostra de pontos da cena é representado por um número.

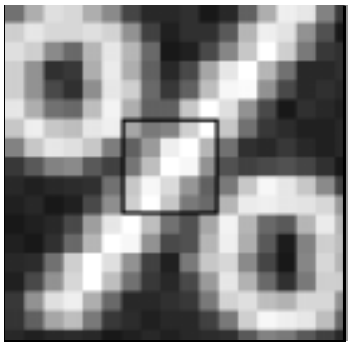

Figura 1.1: Imagem Digital

164	122	131	206	254
98	155	238	254	180
118	188	254	247	147
196	246	216	148	98
205	254	196	122	106

Figura 1.2: Representação Numérica

Quando a representação numérica é feita por números pertencentes a um subconjunto finito dos números inteiros, $\mathbb{Z}$, dizemos que ela é uma **imagem digital**. Quando nesse conjunto estiverem presentes apenas dois valores, por exemplo $\{0, 1\}$ ou $\{0, 255\}$, a imagem digital será chamada de **imagem binária**, caso contrário ela será chamada de **imagem em níveis de cinza** e cada valor será um **nível de cinza**. Este conceito está exemplificado na fig. 1.1 e em uma ampliação da sua parte central mostrada na fig. 1.2.

O grande avanço tecnológico dos últimos tempos, tanto em termos de "hardware" como em termos de "software" na área de computação, possibilitou que o computador fosse cada vez mais utilizado na substituição de tarefas que antes eram realizadas por seres humanos; assim como em novas tarefas que foram surgindo devido as essas novas tecnologias [1].

Devido a isso, verificamos também a utilização cada vez maior do computador nas tarefas de extração de informações de imagens para auxiliar na solução dos mais variados tipos de problemas práticos. Em diversos casos, essa utilização é fundamental, como mostramos em alguns exemplos abaixo:

- **Descoberta automática de fissuras em materiais.** É um trabalho útil para a indústria, principalmente de laminados e estruturas metálicas. As fissuras normalmente são invisíveis a olho nu e as imagens são tomadas utilizando-se radiação de raios-X, que são prejudiciais ao ser humano.

- **Contagem de glóbulos brancos em amostras de sangue.** É útil para laboratórios de análises clínicas e hospitais. Sua realização pelo computador torna o trabalho mais preciso e não desgastante para a vista do examinador.

- **Cálculo do volume sanguíneo nas câmaras do coração.** Esta é uma tarefa usual em exames tomográficos, mas é muito difícil de ser executada sem o auxílio do computador por envolver muitos cálculos sobre várias imagens.

- **Contagem de micro-organismos em amostras orgânicas.** Outra tarefa laboratorial usual em cujo auxílio o computador é útil, pois além de ser desgastante para a vista, pode ser insalubre.

Para que seja possível a utilização de imagens na solução desses problemas, elas devem ser **adquiridas** e **digitalizadas** (**amostradas** e **quantizadas**). Não entraremos nos detalhes destes processos já que existem textos introdutórios muito bons que abordam o assunto [DV84, GW92, Ros69, Pra91, Bax94]. Daqui para frente, neste texto, usaremos a palavra *imagem* para designar a imagem digital.

[1] Essa substituição tem muitas razões e conseqüências sociais (mesmo quando ela é feita da maneira correta, i.e., quando se alivia um ser humano de executar uma tarefa degradante e dá-se a ele uma tarefa mais digna), mas como este não é um trabalho de filosofia, não discutiremos esses problemas aqui.

1.1 Processamento de Imagens

Após a digitalização, podem ser feitas diversas transformações na imagem até conseguir-se a solução do problema para o qual ela está sendo usada. Essas transformações e as técnicas a elas relacionadas podem ser agrupadas em quatro conjuntos bem característicos, quais sejam:

- **Filtragem, Correção e Melhoria da Imagem**: As transformações desse conjunto são úteis para eliminar ruídos indesejáveis originados nos processos de aquisição ou transmissão da imagem. Podem ser usadas para eventuais correções de falhas no processo de aquisição, por exemplo, a rotação da imagem para a correção de sua inclinação. Podem, também, ser usadas para o realce, ou ainda, modificação de características da imagem [GW92, Ros69, Pra91].

- **Análise de Imagens**: Neste conjunto encontram-se diversas transformações e técnicas relacionadas com os processos de separação, representação, reconhecimento e classificação dos objetos que formam a imagem. Uma das etapas fundamentais da análise de imagens é a segmentação (ou separação) da imagem nas partes que a compõe, que é o objeto de estudo deste trabalho [GW92, Ros69, Pra91, Dav90, YC74, TG74, DH73].

- **Compressão**: Uma preocupação constante de quem trabalha com imagens é com o seu armazenamento pois elas ocupam, tipicamente, milhares de "bytes" de memória. Esse problema torna-se realmente grave quando, diariamente, várias imagens têm de ser adquiridas, processadas e armazenadas (como acontece num hospital, por exemplo). Para diminuir o espaço de armazenamento das imagens, existem técnicas e transformações eficientes de compressão que podem reduzir significativamente o espaço que a imagem ocupa, com ou sem perda de informações [GW92, Ros69, Pra91, Hal79, Bax94].

- **Síntese de Imagens**: As transformações e técnicas deste grupo são importantes para a reconstrução geométrica de imagens, por exemplo, a construção da representação volumétrica da imagem de um órgão do corpo humano a partir das suas imagens planas obtidas num exame tomográfico [GW92, Ros69, Pra91, Hal79, Bax94].

A área que estuda todas as técnicas mencionadas acima, e ainda as relacionadas às etapas de aquisição da imagem, chama-se **Processamento de Imagens Digitais (PDI)**. Na figura 1.3, esquematizamos as etapas usuais desta área.

1.2 Segmentação de Imagens

Neste trabalho concentraremos nossa atenção no grupo que chamamos de análise de imagens. Dentro desse grupo estudaremos métodos e transformações úteis para a **segmentação** de imagens. Em processamento de imagens, **segmentar** significa encontrar uma **partição dos pontos da imagem** de forma a separar os objetos de interesse que a compõe. Por exemplo: numa imagem que contenha figuras geométricas, podemos querer separar os pontos que formam círculos dos demais pontos da imagem. Numa imagem de células, podemos querer segmentar as bordas das

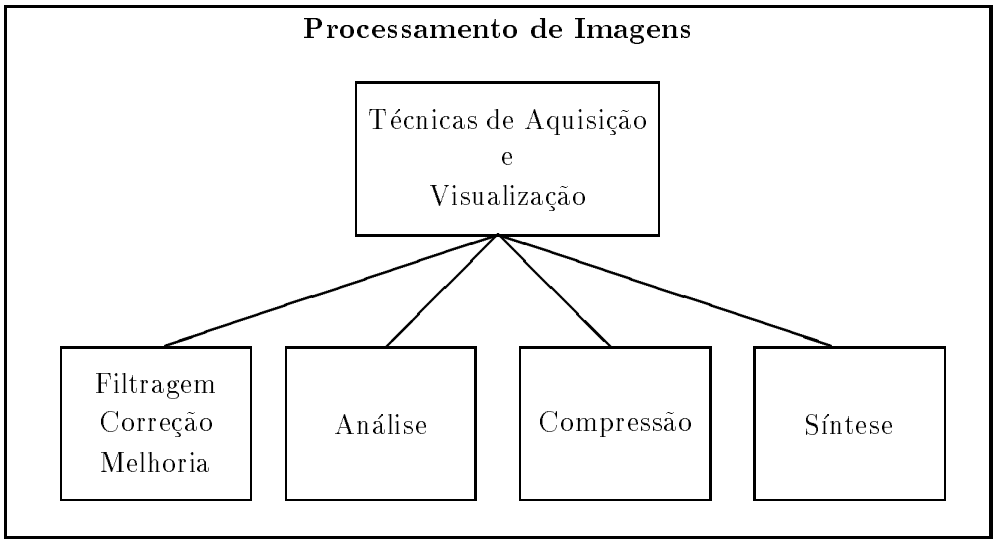

Figura 1.3: Etapas no Processamento de uma imagem

células do resto da imagem. Na imagem de um coração, podemos querer segmentar o ventrículo esquerdo do resto. Numa imagem de um texto, podemos querer segmentar os parágrafos pela forma com que estão alinhados: centralizado, à direita ou à esquerda.

Estes poucos exemplos são suficientes para nos dar a noção do quão importante é conseguir um resultado preciso na segmentação dos objetos de interesse na imagem. É a partir dela que podemos extrair informações da imagem, em geral, medidas dos objetos segmentados.

Nós estamos acostumados com a relativa facilidade e rapidez [2] com que podemos extrair visualmente padrões, objetos ou informações de uma cena, mesmo quando esta não é estática. Isto pode levar-nos a pensar que a automatização da mesma seja igualmente simples, o que não é verdade. Computacionalmente, segmentar uma imagem é uma tarefa não trivial e cujos resultados nem sempre são satisfatórios.

Há diversos problemas que podem impedir o sucesso de uma segmentação como, por exemplo, excesso de ruído na imagem digitalizada, falta de informação suficiente para distinguir as partes da imagem, excesso de objetos na imagem, sobreposição de objetos ou defeitos na imagem a ser segmentada (luminosidade irregular, manchas, inclinação anormal, distorções), etc. No entanto, esse é um assunto muito estudado dada a sua importância para a análise de imagens.

As transformações usadas nos métodos de segmentação de imagens, que hoje em dia já se tornaram clássicos fazem, de modo geral, uso do valor associado a cada ponto na imagem, ou numa vizinhança dele, para: agrupar pontos que possuem características semelhantes e formar regiões com eles; realçar certas regiões características da imagem, por exemplo bordas de objetos da imagem; segmentar apenas os pontos que estejam associados a certos valores, etc.

[2] Desde que as imagens não sejam feitas com o propósito especial de enganar o julgamento que fazemos sobre elas.

Por exemplo, na imagem da Fig. 1.4, usando como característica apenas a luminosidade (intensidade) de cada ponto, individualmente, podemos distinguir visualmente oito regiões diferentes (sete figuras arredondadas e o fundo da imagem). Uma transformação muito utilizada em segmentação, conhecida como limiarização ou "threshold," pode ser aplicada sobre a imagem em questão para separar os quatro objetos mais escuros do resto da imagem. O resultado pode ser visto na figura 1.5

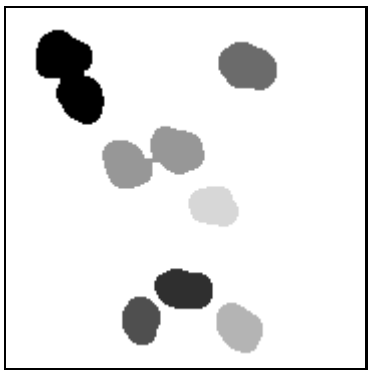

Figura 1.4: Segmentação visual por níveis de cinza

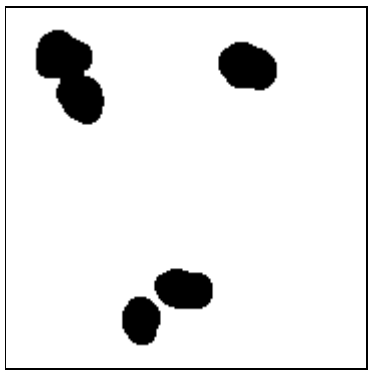

Figura 1.5: Limiarização

Porém, a combinação das transformações existentes e, muitas vezes, a criação de novas transformações para a solução de um problema é feita de uma forma "ad-hoc," i.e., de uma forma específica para a solução do problema cuja característica já conhecemos. Ou seja, o profissional tem de usar sua heurística pessoal baseada na experiência em resolver tais problemas para conseguir um bom resultado, com o agravante que muitas vezes esses métodos e regras pessoais são restritos a um conjunto pequeno de problemas.

Uma outra metodologia, que depende menos da experiência e mais de métodos estatísticos, é a de reconhecimento de padrões. Ela consiste na escolha e posteriormente na extração de alguns tipos de medidas para cada ponto, ou conjunto de pontos, da imagem, para a construção de vetores de atributos. A aplicação de métodos estatísticos sobre esses atributos resulta num critério de decisão para a classificação dos objetos na imagem. Esta técnica pode ser de muita utilidade na segmentação. Além disso, ela é robusta no sentido que podemos aplicar o critério resultante sobre imagens que tenham características semelhantes.

Algumas técnicas de segmentação, consideradas hoje em dia como clássicas, serão apresentadas no capítulo 2 deste trabalho.

1.3 Uma outra abordagem

O objetivo do nosso trabalho, após uma breve introdução sobre os métodos clássicos, é apresentar o problema da segmentação de imagens estudado sob o ponto de vista de uma poderosa teoria algébrica, a **Morfologia Matemática**.

A Morfologia Matemática (*MM*) nasceu em meados dos anos 60 na "École Nationale Superieure des Mines de Paris", a partir das idéias de George Matheron e Jean Serra [BB94]. A idéia inicial deles era extrair informações de imagens binárias a partir de transformações de formas ou texturas usando duas operações elementares as quais chamaram de **dilatação** e **erosão,** baseadas respectivamente nas operações de soma e subtração de Minkowski [Min03] para conjuntos.

A extensão da teoria para tratar as imagens em níveis de cinza foi feita, principalmente, na segunda metade dos anos 70; ou seja, após quase dez anos de pesquisa sobre operadores entre conjuntos [Ser82].

Um grande avanço teórico foi feito, posteriormente, com a descoberta de que tanto o conjunto das imagens binárias, como o conjunto das imagens em níveis de cinza, possuem uma estrutura algébrica comum, a estrutura de reticulado completo, i.e., são conjuntos parcialmente ordenados, que possuem supremo e ínfimo e cada subconjunto deles também possui um supremo e um ínfimo [Bir67].

O paradigma central da *MM* é a descrição dos operadores entre imagens, ou mais geralmente entre reticulados completos, em termos de uma linguagem formal, a *Linguagem Morfológica* ($\mathcal{ML}$), cujo vocabulário são os operadores (**dilatação** e **erosão**) e operações (**negação, supremo** e **ínfimo**) elementares da *MM*.

Esta linguagem é completa, no sentido que ela pode representar qualquer operador entre funções, e expressiva, i.e., muitos operadores úteis podem ser representados com frases curtas da $\mathcal{ML}$. Uma frase da $\mathcal{ML}$ é chamada de um **operador morfológico** [BB92, BB91, BB93, BB94].

Uma implementação da $\mathcal{ML}$ é chamada de *Máquina Morfológica* (**MMach**) e um programa de uma MMach é uma implementação de um operador morfológico nessa máquina [BB92].

A Morfologia Matemática tem ganho cada vez mais espaço no âmbito do processamento de imagens pois todas operações entre imagens podem ser vistas como operações entre reticulados completos. Mais ainda, pode-se provar que qualquer transformação de uma imagem em outra pode ser representada por uma composição de operadores elementares, como veremos no capítulo 3, onde apresentaremos os fundamentos da *MM*. De fato, todas as transformações que apresentaremos no capítulo 2 podem ser fatoradas nos operadores elementares da *MM* e implementadas em uma MMach, o que nos mostra a generalidade da teoria.

1.4 A segmentação sob esta abordagem

No capítulo 4 reapresentaremos o problema de segmentação de imagens utilizando os operadores morfológicos. Daremos ênfase a dois modelos, o "ad hoc", i.e., o modelo formado por técnicas heurísticas baseadas na experiência e no engenho do profissional, e ao modelo de Beucher, que é útil para a extração de contornos dos objetos na imagem. Este último foi introduzido por Serge Beucher [BM93, Beu90, MB90, SV92] e sua importância reside no fato dele facilitar o encontro dos contornos dos objetos na imagem (que é um problema difícil) fazendo com que tenhamos apenas de extrair subconjuntos dos objetos que queremos segmentar, que é um problema menos complexo.

Por exemplo, na Fig. 1.6 vemos a imagem de um bloco de onde queremos segmentar as arestas. Este problema tem uma aplicação prática importante em ambientes que utilizam braços mecânicos robotizados. Ao invés de resolver esse problema diretamente, usando o modelo de Beucher, temos de resolver o problema de encontrar marcadores para cada face que gostaríamos de separar e para o objeto todo, como na Fig. 1.7. Feito isso, o restante do processo é automático. O resultado final pode ser visto na Fig. 1.8.

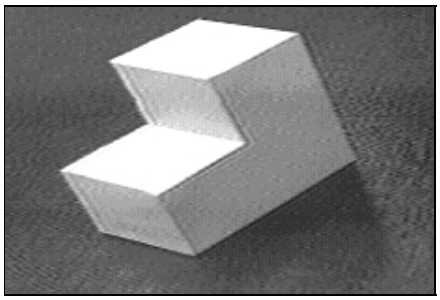

Figura 1.6: Imagem em níveis de cinza

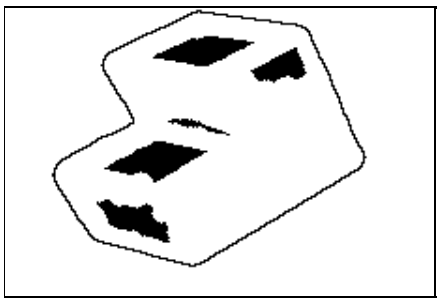

Figura 1.7: Marcadores

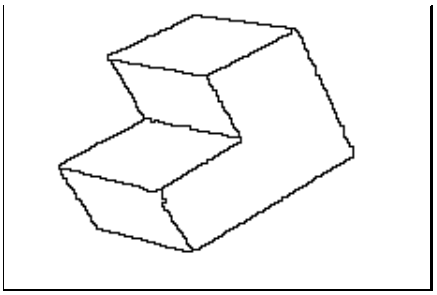

Figura 1.8: Resultado da segmentação

No capítulo 5 apresentaremos alguns problemas práticos de segmentação [3] com os quais trabalhamos e as respectivas soluções que encontramos (algumas originais) usando os operadores da *MM*. Veremos a solução detalhada de problemas como o da Fig. 1.6, mostrando a cada passo da solução as imagens intermediárias.

1.5 Quão eficientes são os algoritmos para isso?

Os operadores elementares da *MM* podem ser implementados facilmente e têm complexidade $\mathcal{O}(n)$, onde n é o número de pontos da imagem. Porém, é comum a utilização de operadores mais

[3] Todo o desenvolvimento prático do trabalho foi feito usando o KHOROS (ver apêndice A) em ambiente SUN (utilizando os recursos computacionais do IME-USP) e em ambiente LINUX (utilizando os recursos computacionais do projeto VERASS, no qual implementamos uma rede integrada à rede Internet e instalamos o KHOROS e a MMACH)

complexos que são formados pela combinação de vários operadores elementares. Nestes casos, a eficiência dos algoritmos pode diminuir consideravelmente se ele não tiver uma implementação mais rápida que aproveite algumas de suas particularidades.

No capítulo 6 apresentaremos algoritmos rápidos que simulam o funcionamento de alguns dos operadores morfológicos mais importantes para a segmentação de imagens, os quais implementamos utilizando o ambiente Khoros (ver apêndice 1) e usamos para resolver vários dos problemas do capítulo 5. O problema de se usar os algoritmos rápidos é que, da forma com que são implementados, muda-se a estrutura da MMach (onde os operadores são construídos utilizando-se os operadores elementares) e isso é ruim tanto se a MMach é implementada por "software" como por "hardware". Por outro lado, se pudéssemos aproveitar algumas idéias desses algoritmos rápidos para melhorar a eficiência das operações e dos operadores elementares, então a estrutura da MMach só seria mudada para essas operações e operadores, e assim poderíamos compô-los para termos operadores complexos mais eficientes.

A busca de algoritmos eficientes coerentes com a estrutura de uma MMach é a motivação do capítulo 7, cuja abordagem é original. Mostraremos como, usando estruturas de dados adequadas e aproveitando as idéias das implementações do capítulo anterior, podemos fazer implementações eficientes para os operadores elementares da *MM*. Uma MMach cujos operadores foram implementados desta maneira é mais rápida e flexível, pois pode-se implementar eficientemente os operadores que motivaram estas implementações dos operadores elementares e de outros que possuam características semelhantes. Apresentaremos ali, também, testes de comparação de eficiência das diversas formas de implementação de alguns algoritmos morfológicos.

Capítulo 2

Segmentação de Imagens

Neste capítulo faremos um apanhado geral dos métodos mais utilizados em segmentação de imagens, que aqui chamamos de **clássicos**, em contraposição aos métodos de segmentação introduzidos pela Morfologia Matemática.

O objetivo disto é criar um vocabulário básico e introduzir as técnicas clássicas mais importantes no âmbito da segmentação de imagens. Não pretendemos com isso fazer um trabalho fechado, no sentido que dispense o estudo de outras referências. Nem pretendemos fazer uma análise comparativa das técnicas para concluir que certa técnica é melhor do que uma outra. Como veremos, a adequação de uma técnica é extremamente dependente do problema que se está resolvendo. Por fim, queremos com este pequeno resumo dar subsídios para que se perceba, ao final do trabalho, que a Morfologia Matemática é uma abordagem equivalente à abordagem clássica.

É raro o processo de extração automática de informações de imagens que não utilize segmentação. Assim, o resultado da segmentação de uma imagem pode determinar o sucesso, ou o fracasso, de toda a análise daquela imagem. É por isso que alguns cuidados gerais que devem ser tomados desde a aquisição da imagem, como por exemplo, utilização de luz com direção e intensidades controladas e "hardware" de aquisição de boa qualidade, tornam-se ainda mais importantes quando se pretende segmentar a imagem.

Existem diversas opiniões a respeito do que é segmentar uma imagem [HS85, CA79, Kan80, RD79, YB89, Hal79, GW92, Pra91], mas não existe nenhuma teoria sobre o assunto. Uma conseqüência disto é que existem muitas técnicas operacionais e computacionais para resolver os diversos problemas desta área. No entanto, não é difícil depararmos com casos em que a imagem não possui informações suficientes para ser segmentada sem o uso de um conhecimento prévio a respeito dos objetos que estão sendo segmentados [Mar82]. Um exemplo de uma imagem que contém objetos assim é a vista na fig. 2.1. As ampliações desta imagem (fig. 2.2 e fig. 2.3) mostram duas células na parte superior que estão juntas e só sabemos que são duas comparando com o formato e tamanho das outras células na imagem. Também, é fácil encontrar um problema de segmentação para o qual os métodos clássicos existentes até o momento não sejam satisfatórios para resolvê-lo.

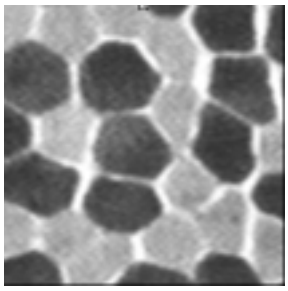

Figura 2.1: Fronteira duvidosa

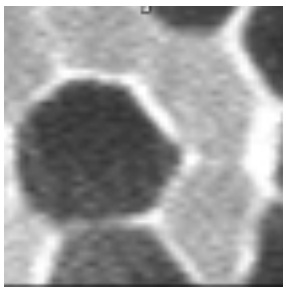

Figura 2.2: Primeira ampliação

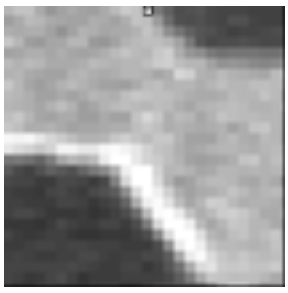

Figura 2.3: Segunda ampliação

Antes de apresentar os métodos de segmentação, vamos estabelecer algumas notações úteis que usaremos daqui para frente.

2.1 Notações Básicas

Seja $\mathbb{E}$ um conjunto de pontos no plano[1], onde os pontos são dados pelas suas coordenadas. Por questões práticas, o conjunto $\mathbb{E}$ é normalmente definido como um subconjunto retangular de $\mathbb{Z} \times \mathbb{Z}$, por exemplo, $[a, b] \times [c, d] \subset \mathbb{Z} \times \mathbb{Z}$.

Um **grafo de conectividade**[2] [Ser82, RP66, KR89] G é um par ordenado de conjuntos $(\mathbf{E}, D)$, onde $\mathbb{E}$ é um conjunto de pontos e D é um subconjunto de pares não ordenados de pontos de $\mathbf{E}$, denominado conjunto de arestas. Uma aresta $\{x, y\} \in D$ junta os pontos x e y e define uma relação que denominaremos de **vizinhança** entre x e y. Isto é, se $\{x, y\} \in D$, então x e y são pontos **vizinhos** [Ser82, RP66, KR89].

Se (g, h) são as coordenadas de um ponto qualquer $x \in \mathbb{E}$ e se, segundo G, x tem como vizinhos os pontos $(g+1, h), (g-1, h), (g, h+1)$ e $(g, h-1)$ (fig. 2.4) então $\mathbf{E}$ é dito ser 4-adjacente (fig. 2.5).

[1] Restringiremos nosso estudo aos conjuntos definidos em espaços de uma ou duas dimensões, denotados por 1D e 2D, respectivamente. A extensão de vários dos conceitos e definições encontrados neste trabalho para espaços de dimensão maior não é difícil.

[2] também conhecido como grade de conectividade

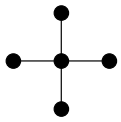

Figura 2.4: Um ponto e seus 4 vizinhos

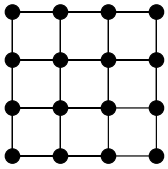

Figura 2.5: Espaço 4-adjacente

Se além desses vizinhos, x tem como vizinhos os pontos $(g+1, h+1), (g+1, h-1), (g-1, h+1)$ e $(g-1, h-1)$ (fig. 2.6), então $\mathbb{E}$ é dito ser 8-adjacente (fig. 2.7).

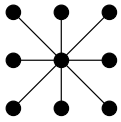

Figura 2.6: Um ponto e seus 8 vizinhos

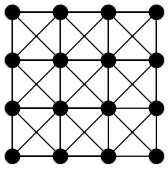

Figura 2.7: Espaço 8-adjacente

Dependendo de como $\mathbb{E}$ for definido, nem todos os vizinhos de um ponto $x \in \mathbb{E}$ precisam existir. Por exemplo, se $\mathbb{E}$ for definido como o subconjunto $[a, b] \times [c, d] \subset \mathbb{Z} \times \mathbb{Z}$, então nem todos os vizinhos de x existem se $x = (g, h)$, quando $g = a$ ou $g = b$ ou $h = c$ ou $h = d$.

Outros tipos de grades de conectividade podem ser definidas sobre os pontos de $\mathbb{E}$ [KR89, Ser82]. Por exemplo, um tipo muito utilizado na "École Nationale Superieure des Mines de Paris" é a grade 6-adjacente (fig. 2.9, onde cada ponto tem 6 vizinhos (fig. 2.8). Neste trabalho vamos nos restringir apenas aos conjuntos $\mathbb{E}$ 4-adjacentes ou 8-adjacentes.

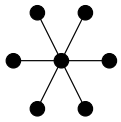

Figura 2.8: Um ponto e seus 6 vizinhos

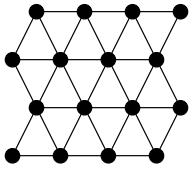

Figura 2.9: Espaço 6-adjacente

Sejam p e q dois pontos de $\mathbb{E}$. Um **caminho** C de p a q é uma seqüência de pontos de $\mathbb{E}$, $C = \{p_0, p_1, \ldots, p_n\}$, tal que os extremos de C, p_0 e p_n são, respectivamente, os pontos p e q, além disso, p_i é vizinho de $p_{i+1}, \forall i \in [0, n-1]$.

O comprimento do caminho C de p a q é igual ao número de pontos da seqüência menos um, $|C| = n - 1$.

Uma maneira de definir a **distância** entre p e q, denotada $d(p, q)$, é o comprimento do menor

caminho de p a q. Existem outras maneiras de definir a distância entre dois pontos de $\mathbb{E}$ [Ser82], mas não as veremos aqui.

Um subconjunto, ou **região**, $X \subset \mathbb{E}$ é conexo[3], segundo G, se $\forall p, q \in X$ existe um caminho de p a q consistindo inteiramente de pontos contidos em X.

Seja $x \in \mathbb{E}$, $X \subset \mathbb{E}$ e $d(x, y)$ uma distância. A distância entre o ponto x e um subconjunto X, sob d, é dado pelo valor:

$$d(x, X) = min\{d(x, y) : y \in X\}$$

Um ponto de $\mathbb{E}$ que esteja associado ao valor 0 na imagem é um **ponto preto** ou **ponto de fundo**. Um ponto de $\mathbb{E}$ que esteja associado a um valor diferente de 0 na imagem é um **ponto branco**. O conjunto de todos os pontos pretos da imagem formam o fundo da imagem.

Define-se a **borda interior**, ∂X, de um conjunto $X \subset \mathbb{E}$, como sendo o conjunto dos pontos $p \in X$ tal que p tem pelo menos um vizinho que não pertence a X.

Duas regiões X e Y são **adjacentes** se existem $x \in X$ e $y \in Y$, tal que x e y são vizinhos.

O **complemento** de uma região $X \subset \mathbb{E}$ é denotado por $\overline{X}$.

Dada uma região conexa $X \subset \mathbb{E}$, um **buraco** na região X é uma região $Y \subset \mathbb{E}$, adjacente a X, tal que não existe um caminho $C \subset \mathbb{E}$, totalmente contido em $\overline{X}$, que ligue um ponto $p \in Y$ com um ponto $q \in \overline{X}/Y$.

Denota-se por $K^{\mathbb{E}}$ o conjunto de todas as funções $f : \mathbb{E} \to K$, onde $K = [0, k]$ é um subintervalo fechado do conjunto $\mathbb{Z}$.

Dessa forma, uma imagem em níveis de cinza pode ser representada por uma função[4] $f \in [0, k]^{\mathbb{E}}$, onde $k \in \mathbb{Z}$, e $k > 1$.

Da mesma forma, uma imagem binária, i.e., uma imagem na qual cada ponto só pode assumir dois valores 0 e k pode ser representada por uma função $f \in \{0, k\}^{\mathbb{E}}$.

Se os conjuntos ${K_1}^{\mathbb{E}}$ e ${K_2}^{\mathbb{E}}$ representam conjuntos de imagens, um operador $\Psi : {K_1}^{\mathbb{E}} \to {K_2}^{\mathbb{E}}$ será chamado de um **operador entre imagens**.

Definição 2.1.1 *Seja $f \in K^{\mathbb{E}}$ uma imagem definida em $\mathbb{E}$ e P um predicado lógico de segunda ordem [HDW94] sobre $X \subset \mathbb{E}$.* **Segmentar** *a imagem f [GW92, Har92] é encontrar uma partição $\Omega = \mathbb{E}_1, \mathbb{E}_2, \ldots, \mathbb{E}_n$ de $\mathbb{E}$ tal que:*

1. $\bigcup_{i=1}^{n} \mathbb{E}_i = \mathbb{E}$
2. $\mathbb{E}_i \cap \mathbb{E}_j = \emptyset \quad \forall i, j \in [1, n], i \neq j$
3. $P(\mathbb{E}_i)$ *é verdadeiro* $\forall i \in [1, n]$

[3] A conexidade por caminhos é uma possível definição de conexidade para subconjuntos de $\mathbb{E}$, mas existe uma outra definição topológica que não depende da definição de caminho, apenas da definição de adjacência, veja [KR89]

[4] A constante 0 provém do fato dela estar associada a um dos níveis de cinza da imagem, em geral, o preto.

4. $P(\mathbb{E}_i \cup \mathbb{E}_j)$ *é falso,* $\forall i, j \in [1, n]$, $(i \neq j)$

As condições 1 e 2 definem uma partição usual de um conjunto. Em 1 assegura-se que todo ponto da imagem pertence a algum elemento da partição, o que é razoável pois uma segmentação deve ser completa. Em 2 garante-se que nenhum ponto da imagem pode estar em mais de um elemento da partição ao mesmo tempo, ou ainda, nenhum ponto pode satisfazer a critérios diferentes ao mesmo tempo.

As condições 3 e 4 especificam Ω no que tange à segmentação. Em 3 impõe-se que os pontos de cada elemento $\mathbb{E}_i$ da partição Ω satisfaçam o predicado lógico P, i.e., que cada $\mathbb{E}_i$ satisfaça a algum critério definido pelo predicado. Note que P é um predicado definido sobre conjuntos, mais especificamente sobre regiões conexas, e não sobre um certo elemento do conjunto, i.e., é um predicado de segunda ordem, como em Lógica costuma-se chamar. Em 4 impõe-se que se tomarmos a união de duas regiões distintas, $\mathbb{E}_i \cup \mathbb{E}_j$, então $\mathbb{E}_i$ e $\mathbb{E}_j$ têm de ser diferentes com relação ao predicado P. Por exemplo, tomemos uma imagem com objetos de diversas cores. Vamos supor que queremos segmentar os objetos azuis e os objetos vermelhos da imagem. Seja P o predicado: (Os pontos de $\mathbb{E}_i$ são todos vermelhos) ou (Os pontos de $\mathbb{E}_i$ são todos azuis) ou (Os pontos de $\mathbb{E}_i$ são diferentes de vermelho e azul), onde Ω é uma partição de $\mathbb{E}$ e $\mathbb{E}_i \in \Omega$. Uma partição $\mathbb{E}_1, \mathbb{E}_2$ e $\mathbb{E}_3$, tal que $\mathbb{E}_1$ seja formado por todos os de pontos vermelhos da imagem, $\mathbb{E}_2$ por todos os pontos azuis da imagem e $\mathbb{E}_3$ por todos os outros pontos da imagem, satisfaz 1 pois contém todos os pontos da imagem, satisfaz 2 pois nenhum ponto pode ter duas cores simultaneamente, satisfaz 3 pois o predicado é verdadeiro para cada $\mathbb{E}_i$ e satisfaz 4 pois se unirmos quaisquer dos elementos da partição teremos um novo elemento com pontos de pelo menos uma das cores, azul ou vermelho, o que torna falso o predicado para esse elemento.

Pela definição 2.1.1, dada uma imagem e um certo problema de segmentação temos dois problemas a resolver:

- Formular um predicado lógico que possibilite através das informações disponíveis na imagem a solução do problema.

- Achar um operador entre imagens, que pode ser composto por vários outros, que particione o espaço $\mathbb{E}$ segundo as regras do predicado, de preferência eficientemente.

Existem várias maneiras de encontrar esses operadores. A maior parte baseia-se nas descontinuidades, ou semelhanças, entre os níveis de cinza dos pontos da imagem. Outros baseiam-se em características como, por exemplo, cor, tamanho, forma, textura e etc. Vamos ver a seguir algumas técnicas que já se tornaram de grande importância em Processamento de Imagens e que, por isso, são conhecidas como clássicas.

Trataremos na maioria das vezes com imagens de duas dimensões, mas a teoria aplica-se muitas vezes a imagens de qualquer dimensão inteira.

2.2 Métodos Clássicos de Segmentação

Vamos mostrar a seguir um pequeno resumo de algumas das técnicas e métodos clássicos existentes, começando com as que se baseiam em similaridades, como o "threshold"[5] e passando para as que se baseiam em dissimilhanças, como o gradiente.

2.2.1 Threshold

Um dos operadores mais utilizados para segmentação de objetos que possuem características semelhantes em termos de luminosidade é o **threshold**,

$$T : K_1^{\mathbf{E}} \longrightarrow \{0, k_2\}^{\mathbf{E}}.$$

Dada uma imagem $f \in K_1^{\mathbf{E}}$, o objetivo deste operador é atribuir o mesmo rótulo ($k_2 \in \mathbb{Z}$) para todos os pontos $p_i \in \mathbb{E}$ tais que $f(p_i)$ satisfaça a um certo critério.

Threshold Simples

Existem vários tipos de threshold, o mais simples, e de maior uso, depende de apenas um parâmetro h, normalmente chamado de **limiar** ou mesmo de **threshold**, definido por:

$$T_h(f)(x) = \begin{cases} k & \text{se} \quad f(x) \leq h \\ 0 & c.c. \end{cases}$$

É fácil ver que o operador threshold não é inversível e que neste caso seu resultado é uma imagem binária que vale k apenas nos pontos onde o valor da função f é menor que h.

Threshold Múltiplo

Este operador é chamado de **threshold múltiplo** pois admite dois limiares h_1 e h_2, e é definido por:

$$T_{h_1,h_2}(f)(x) = \begin{cases} k & \text{se} \quad h_1 \leq f(x) \leq h_2 \\ 0 & c.c. \end{cases}$$

No caso que $h_1 = 0$, o threshold múltiplo é equivalente ao threshold simples.

A imagem da fig. 2.10, mostra um texto digitalizado e as imagens fig. 2.11, fig. 2.12, fig. 2.13, fig. 2.14 e fig. 2.15 mostram diversos exemplos do resultado do operador threshold múltiplo aplicado sobre esse texto[6]. Os parâmetros utilizados no threshold (h_1 e h_2) estão grafados nas legendas das figuras, com exceção das vezes que $h_1 = 0$; nestes casos ele não aparece.

[5] Neste texto, todas as palavras em um idioma estrangeiro serão grafadas entre aspas " ", entretanto, esta palavra será utilizada daqui para frente sem as aspas, por ser muito comum.

[6] Devido ao tipo de impressão utilizado neste trabalho, não é fácil perceber que as letras na imagem em níveis

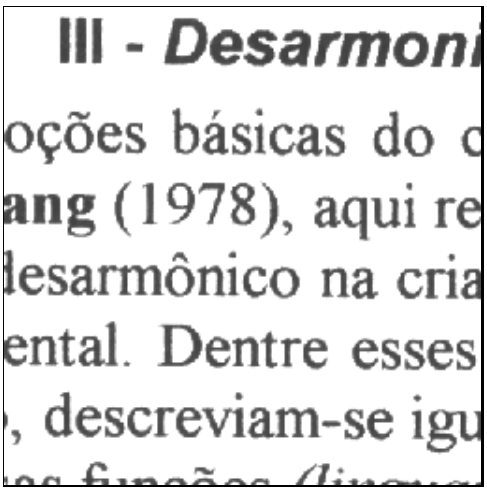

Figura 2.10: Imagem extraída de um texto

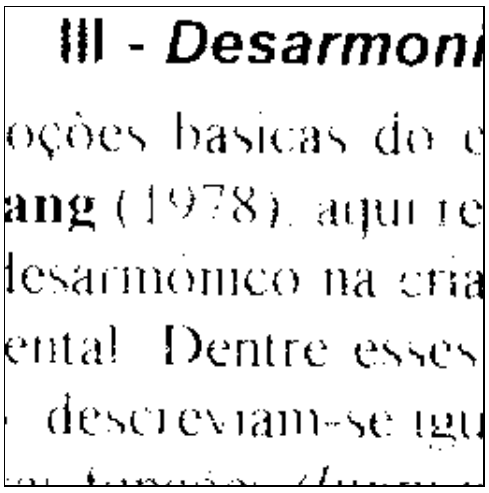

Figura 2.11: Parâmetro $h_2 = 85$

Normalmente é difícil encontrar os parâmetros h_1 e h_2 que se ajustem corretamente para toda a imagem nessa variante. No entanto, o operador é útil quando queremos segmentar regiões cujos pontos estejam em uma determinada faixa de intensidades.

Threshold Adaptativo

O operador de **threshold adaptativo** (Adaptative Threshold [Wat72]) é uma generalização do anterior. A idéia aqui é achar parâmetros diferentes para cada parte da imagem, i.e., achar duas funções f_1 e $f_2 \in K^{\mathbf{E}}$, $f_1(x) < f_2(x) \forall x \in \mathbb{E}$, que sirvam de limiares. A fórmula geral é dada por:

$$T_{f_1,f_2}(f)(x) = \begin{cases} k & \text{se} \quad f_1(x) \leq f(x) \leq f_2(x) \\ 0 & c.c. \end{cases}$$

Critérios para o estabelecimento de limiares

Uma maneira para estabelecer limiares para a segmentação de imagens é a análise visual dos níveis de cinza dos pontos dos objetos que queremos segmentar. Por exemplo, a maioria dos pontos da imagem da fig. 2.10 correspondentes às letras têm níveis de cinza abaixo de 150. Um threshold simples com limiar $h = 150$ é suficiente para segmentar todos os caracteres da imagem.

O uso criterioso de operadores de threshold requer, normalmente, o cálculo do histograma[7] dos pontos da imagem de acordo com os seus níveis de cinza para o encontro dos limiares corretos.

---

de cinza têm o interior bem escuro e a intensidade dos níveis vai aumentando em direção às bordas, i.e., as bordas das letras são mais claras.

[7] O histograma dos níveis de cinza de uma imagem é um gráfico cujo eixo das abscissas representa os níveis de cinza da imagem e o eixo das ordenadas representa a freqüência com que cada nível aparece na imagem.

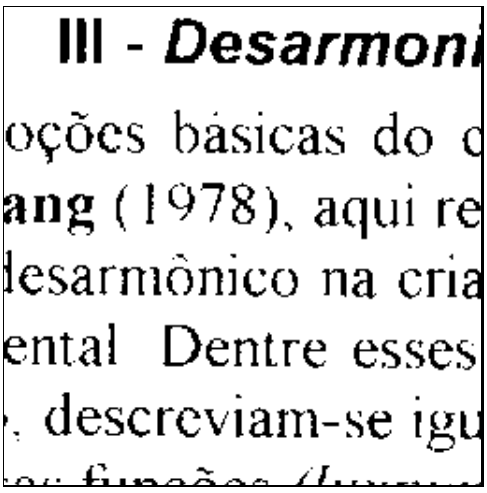

Figura 2.12: Parâmetro $h_2 = 100$

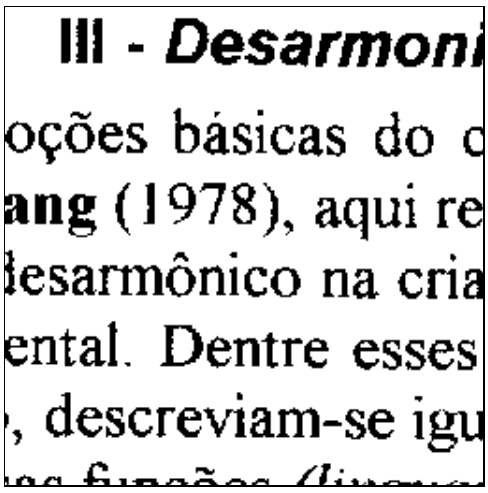

Figura 2.13: Parâmetro $h_2 = 150$

Isto torna a segmentação mais precisa e automática do que quando o mesmo processo é feito por uma amostragem a partir da inspeção visual da imagem.

Quando o histograma da imagem possui um vale como o da figura 2.16, pode-se escolher o limiar como sendo o valor mínimo da curva do histograma. Mas, existem técnicas mais elaboradas para o encontro de limiares [Rus91].

O histograma em conjunto com a limiarização não resolve todos os problemas de segmentação pois não leva em consideração, por exemplo, a forma dos objetos na imagem, i.e., dois objetos de formatos diferentes podem ser indistinguíveis usando-se esta técnica.

Os problemas do mundo real são bem mais complexos e, normalmente, o threshold é usado em conjunto com outros operadores. Além disso, ele é reservado como um dos últimos operadores a se aplicar na imagem, dada a grande perda de informações que acontece após a sua aplicação.

Finalmente, é importante mencionar que os algoritmos que implementam tanto o operador de threshold como o cálculo do histograma têm complexidade linear com relação ao número de pontos da imagem.

2.3 Crescimento de Regiões

Seja $\{\mathbf{E}_i\}$ uma família de subconjuntos de $\mathbf{E}$ cujos pontos são escolhidos de acordo com a imagem $f \in K^{\mathbf{E}}$ a ser segmentada. Inicialmente os subconjuntos são denominados **sementes**, pois é a partir deles que começa o processo. Após o início eles são chamados de **regiões**. Crescimento de regiões é o nome que se dá à técnica de aumentar o número de pontos de cada semente, unindo pontos a ela segundo algum critério de similaridade. O processo termina quando não houver mais

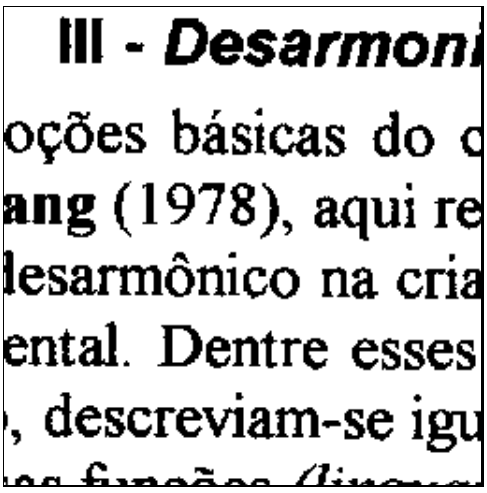

Figura 2.14: Parâmetro $h_2 = 200$

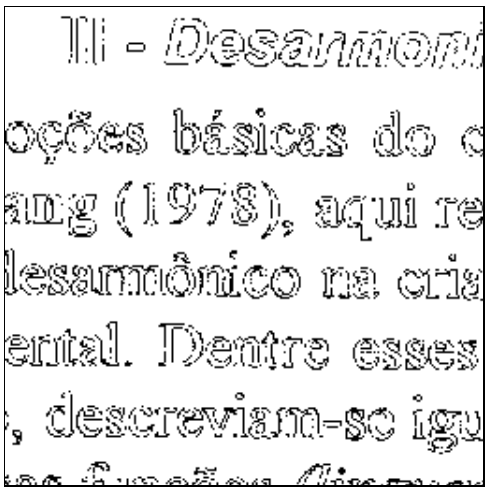

Figura 2.15: Parâmetros $h_1 = 100$ e $h_2 = 200$

pontos em $\mathbf{E}$ que satisfaçam o critério de similaridade. O conjunto das regiões após o término do processo, mais o seu complemento em relação a $\mathbf{E}$, formam uma partição desse espaço.

O crescimento das regiões se faz a partir dos pontos vizinhos da borda das regiões e de um certo critério de semelhança cuja forma mais simples é a diferença absoluta entre os níveis de cinza dos pontos da borda das regiões e seus vizinhos [Har92, HS85].

Na literatura encontramos vários outros critérios para formar regiões e cada um deles resulta em um operador diferente entre funções. Os operadores simples, i.e., baseados apenas num critério de vizinhança, podem dar resultados ruins com maior freqüência, pois uma região A pode ligar-se a uma outra região B devido apenas à diferença existente entre um ponto da borda da região A e seu vizinho na região B.

Uma forma de agrupamento que evita esse problema foi introduzida por Brice e Fenema [BF70]. A técnica pode ser resumida assim:

- Na primeira etapa do processo, pares de pontos vizinhos são ligados para formar um conjunto, que os autores chamaram de **regiões atômicas**, se eles têm o mesmo valor de cinza. Em outras palavras, se eles têm a mesma intensidade e são vizinhos, segundo a grade G, então eles irão formando regiões conexas.

- Na segunda etapa, que eles chamaram de **heurística da fagocitose**, regiões maiores devem absorver as regiões menores segundo o seguinte critério:

  Sejam P_1 e P_2 os comprimentos das bordas das regiões $R_1 \subset \mathbf{E}$ e $R_2 \subset \mathbf{E}$, respectivamente (fig. 2.17). Seja $|I|$ o comprimento da fronteira I entre R_1 e R_2. Seja $|W|$ o comprimento das partes "fracas" da fronteira, i.e., o comprimento de $W \subset I$ onde a diferença absoluta entre os níveis de cinza de R_1 e R_2 através da fronteira é menor do que um certo ϵ_1. Desta forma, duas regiões, R_1 e R_2, são ligadas se:

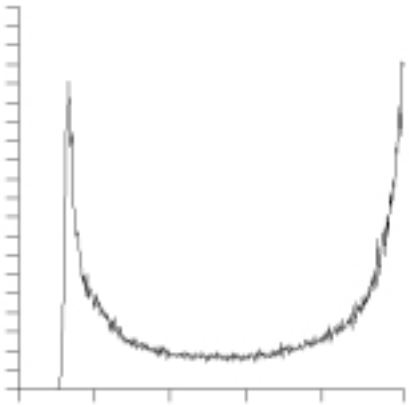

Figura 2.16: Histograma de uma imagem em níveis de cinza

$$\frac{|W|}{\min\{|P_1|,|P_2|\}} > \epsilon_2 \text{ onde } \epsilon_2 = \frac{1}{2}, \text{ usualmente.}$$

Se ϵ_2 é pequeno, duas regiões podem juntar-se com facilidade. Se, ao contrário, ele é grande, só quando uma região praticamente envolver a outra é que elas serão agrupadas.

- Na terceira etapa, regiões adjacentes cuja fronteira é fraca, i.e., têm vários pontos cuja diferença absoluta entre os níveis de cinza através da fronteira é menor do que ϵ_1, são ligadas segundo o seguinte critério:

$$\frac{|W|}{|I|} > \epsilon_3, \text{ onde } \epsilon_3 = \frac{3}{4}, \text{ usualmente.}$$

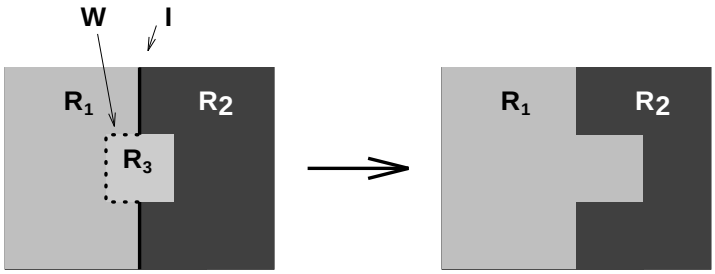

Figura 2.17: Dinâmica do Crescimento de Regiões

Esta técnica de Brice e Fenena produz resultados satisfatórios, segundo a literatura clássica, desde que as imagens não sejam muito complexas, i.e., não tenham muitos objetos a serem segmentados e possuam pouca textura.

2.4 "Split & Merge"

O método de "Split & Merge" (**S&M**) é a composição dos métodos de separar ("Split") e de juntar ("Merge"). O primeiro consiste em, a partir da imagem inteira, ir separando os pedaços em

quartos (quartas partes) se eles não forem suficientemente homogêneos. Este processo é feito até um certo número de regiões pré-estabelecido, ou até que não seja mais possível separá-las (devido à região ter ficado menor que um certo limite ou se tornado um ponto). O segundo consiste em, a partir de uma imagem dividida em pedaços, ir juntando as regiões que forem parecidas.

Para controlar eficientemente este processo, é usada uma estrutura de dados do tipo árvore "quadtree" (i.e., uma árvore na qual cada nó que não seja folha têm quatro filhos).

A imagem inteira é a raiz da árvore, os descendentes são obtidos separando ("Split") a imagem em quatro partes, sucessivamente; os ascendentes são obtidos juntando-se ("Merge") quatro nós da árvore (não necessariamente do mesmo pai) de acordo com os critérios de homogeneidade estabelecidos.

As técnicas de **S&M** podem ser computacionalmente "pesadas" se o processo de separação for iniciado pela imagem inteira. Por outro lado, se o processo começar com a árvore excessivamente separada (ramificada), tal que as folhas são os pontos da imagem, podem ocorrer erros devido às medidas da uniformidade de uma região estarem baseadas, inicialmente, apenas em pontos vizinhos. Assim, o **S&M** inicia, normalmente, num estágio intermediário entre estes dois extremos.

Vários autores relatam também que as imagens segmentadas por este método podem ter segmentos com formato ligeiramente retangulares, devido às regras de divisão e junção [Pra91, Har92, Hal79].

2.5 "Clustering" ou Aglomeração

Uma técnica semelhante à de crescimento de regiões é o "**clustering**", ou **aglomeração** [DH73, Pra91, DV84]. A diferença principal entre essas técnicas é que aquela (crescimento de regiões) atua sobre o espaço de pontos da imagem, i.e., o espaço $\mathbb{E}$, e esta atua sobre um espaço conhecido como **espaço de atributos**.

Atributo é o nome que se costuma dar a uma medida feita sobre uma imagem. Normalmente, ele é relativo às propriedades ou às medidas de determinados conjuntos de padrões. Essas medidas podem ser relativas a tamanho de objetos na imagem, desvio padrão ou média dos níveis de cinza da imagem ou de regiões da imagem, número de objetos em uma certa região, níveis de cinza de vários espectros de luz, componentes de um sistema de cores (RGB, HSB, etc) e etc.

A imagem da fig. 2.18 mostra um exemplo de como coletar um atributo usando uma janela de tamanho 1×3 pontos com centro no ponto do meio. Se p for o ponto do centro da janela, um exemplo de atributo no ponto p é a média dos níveis de cinza dos pontos vistos pela janela.

Um outro exemplo de atributos que podem ser tomados de maneira semelhante são os níveis de cinza dos pontos da imagem vistos por uma janela quando centrada no ponto p. No caso acima, para cada ponto p da imagem teremos 3 atributos (W_1, W_2 *e* W_3), cujos valores são os níveis de cinza dos pontos da janela.

Um espaço de atributos é um sistema cartesiano em que cada eixo representa um atributo diferente. Dada uma imagem, toma-se um vetor de atributos $\mathbf{u} = (u_1, u_2, u_3, ..., u_n)$ para cada

ponto da imagem.

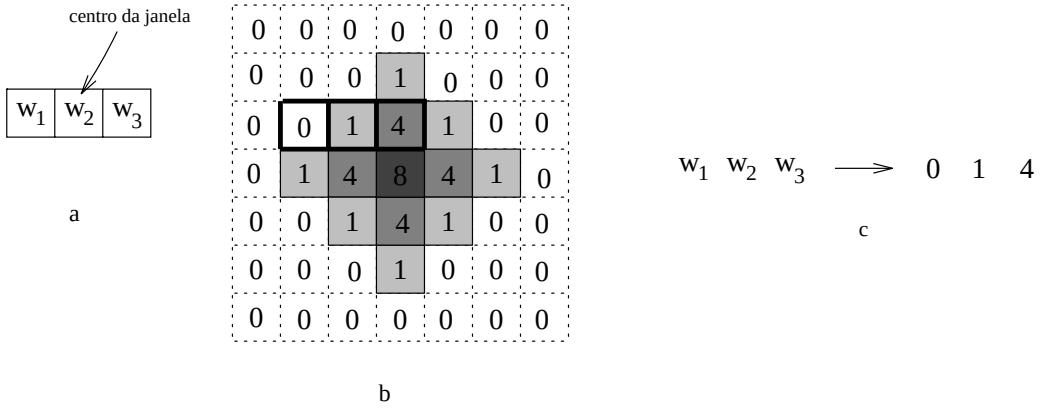

Figura 2.18: Dinâmica da coleta de configurações

No espaço dos atributos vão estar representados N pontos, $X_1, X_2, ..., X_N$. O problema de aglomeração é um problema de particionamento do espaço de atributos em um número K de subconjuntos disjuntos. A idéia por detrás da técnica é que os pontos de um certo objeto ou padrão da imagem devem ter características semelhantes e o conjunto desses pontos formariam, então, um aglomerado ("cluster"), como mostrado na fig. 2.19. Há várias formas de fazer esta aglomeração dos pontos[DH73]; a mais usada parte de um número pequeno de "clusters" (cujos centros são escolhidos dentre os pontos da imagem) e a cada etapa o número de centros vai aumentando com o objetivo de diminuir as distâncias[8] entre os pontos do aglomerado e os novos centros de aglomeração.

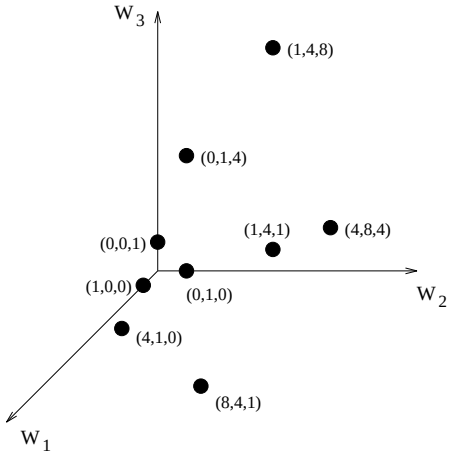

Figura 2.19: Representação de aglomerados em um espaço de atributos

[8] A distância neste caso pode ser a distância usual do espaço cartesiano.

Um dos principais problemas do método é saber o número ótimo de "clusters" para que o resultado da segmentação seja bom. Vamos apresentar a seguir uma técnica de dois estágios, desenvolvida por Coleman e Andrews [CA79], que resolve esse problema. No primeiro estágio são computados os vetores de atributos. No segundo estágio, é determinado o número ótimo de núcleos, seus centros e os pontos que os compõem, segundo um critério de proximidade dos centros de aglomeração.

O algoritmo usado para isso tenta, primeiramente, aglomerar todos os N vetores em torno de 2 centros (que podem ser escolhidos ao acaso dentre os N vetores), segundo a distância do vetor ao centro mais próximo dele. Após isso, o número de centros vai aumentando de 1 e um fator de qualidade β vai sendo computado a cada iteração até que ele atinja um valor máximo (que será o número ótimo de aglomerados). A cada próxima etapa, o centro do novo aglomerado a ser criado será o ponto que possui a maior distância ao centro do aglomerado, dentre os aglomerados atuais.

O cálculo do fator de qualidade β é feito da seguinte forma. Sejam

- N o número de pontos a serem aglomerados
- K o número de aglomerados atual
- M_k o número de pontos do aglomerado A_k (k-ésimo aglomerado)
- $\mathbf{u}_i$ um vetor de características, $1 \leq i \leq N$
- $\mathbf{u}_0$ o vetor médio de todos os vetores, dado por $\mathbf{u}_0 = \frac{1}{M} \sum_{i=1}^{M} \mathbf{u}_i$
- $\mathbf{u}_k$ a média dos vetores de A_k
- S_k o conjunto dos elementos de A_k
- $\mathbf{S_B}$ e $\mathbf{S_W}$, duas matrizes esparsas (externa e interna, respectivamente) dadas por:

$$\mathbf{S_W} = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{M_k} \sum_{x_i \in S_k} [\mathbf{u}_i - \mathbf{u}_k][\mathbf{u}_i - \mathbf{u}_k]^T$$

$$\mathbf{S_B} = \frac{1}{K} \sum_{k=1}^{K} [\mathbf{u}_k - \mathbf{u}_0][\mathbf{u}_k - \mathbf{u}_0]^T,$$

onde u^T é o vetor transposto de u. Finalmente,

$$\beta = tr[\mathbf{S_W}]tr[\mathbf{S_B}],$$

onde tr é o traço da matriz.

Após aglomerados, os diversos "clusters" formam segmentos da imagem.

O "clustering" é uma conhecida técnica de reconhecimento de padrões, mais especificamente, uma técnica de "aprendizado" não supervisionado[DH73] que é usada para a solução de diversos problemas em Processamento de Imagens. A parte heurística de um processo de segmentação por "clustering" consiste na escolha das medidas que serão feitas na imagem. O restante do processo é, geralmente, estatístico.

Até agora vimos as técnicas clássicas baseadas em semelhanças entre os objetos ou padrões da imagem. A partir de agora veremos as técnicas clássicas que são baseadas nas dissimilhanças entre eles. Os objetos que podem ser localizados por estas técnicas podem ser de três tipos: pontos, linhas e arestas.

Os operadores usados neste caso são, normalmente, definidos por uma operação de vizinhança. Seja $\mathbf{N_G(x)}$ o conjunto dos pontos vizinhos do ponto x segundo o grafo de conectividade G. Vamos apresentar os operadores para o caso em que $\mathbf{E}$ é 8-adjacente[9] e vamos dar índices a esses pontos segundo a imagem da fig. 2.20. Os operadores são do tipo (seja $f \in K^{\mathbf{E}}$):

$$\Psi(f)(x) = \sum_{i=1}^{9} w_i f(x_i) \tag{2.1}$$

onde $x_5 = x$, $x_i \in \mathbf{N_G(x)}$ $(i \neq 5)$ e w é um vetor de pesos dados aos pontos para diferenciá-los. O conjunto dos pesos w_i também é conhecido como **máscara detectora** e pode ser representado por uma matriz 3×3 do tipo:

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figura 2.20: Máscara

2.6 Detecção de pontos isolados

O conceito de ponto isolado em uma imagem binária é bem simples e a sua segmentação, mais ainda. Chamamos de **ponto isolado** um ponto $p \in \mathbf{E}$ tal que $f(p) = 1$ e $f(q) = 0, \forall q \in \mathbf{N_G(p)}$.

Para segmentar um ponto isolado, primeiramente usa-se um operador local que o realce em relação aos outros. O operador definido pela equação 2.1 com os pesos w_i dados por $w_5 = 8$ e $w_i = -1$, $1 \leq i \leq 9, i \neq 5$, aplicado à imagem binária $f(x) \in [0,1]^{\mathbf{E}}$ resulta em uma imagem em níveis de cinza onde todo ponto isolado vale 8 (i.e., $\Psi(f)(x) = 8$, caso x seja um ponto isolado) e $\Psi(f)(x) < 8$, caso contrário.

[9] A adaptação desses operadores para outras conectividades pode ser realizada facilmente.

Para completar a segmentação destes pontos, basta aplicar um threshold que segmente os pontos cujo valor seja igual a 8.

Pontos isolados em imagens de níveis de cinza tem uma conceituação um pouco diferente; dizemos que um ponto é isolado quando seu valor difere consideravelmente de seus vizinhos e não, simplesmente, quando todos os seus vizinhos são nulos. O operador que será aplicado à imagem pode ser o mesmo que é aplicado para imagens binárias, mas o threshold dependerá da diferença de níveis de cinza entre o ponto e seus vizinhos que considerarmos suficiente para tratá-lo como isolado.

2.7 Detecção de retas

Da mesma forma que um ponto isolado pode ser segmentado por um operador do tipo definido pela equação 2.1, também pode ser um segmento de reta, desde que o vetor de pesos usado seja adequado para realçar as partes de uma reta. Para cada direção possível das retas que se deseja realçar na imagem deve-se considerar um vetor de pesos para realçá-la. Por exemplo, utilizando a representação de matrizes para os pesos w_i, se quisermos realçar uma reta horizontal, devemos usar a matriz da figura 2.21 e para realçar uma reta a 45^o devemos usar a matriz da figura 2.22.

-1	-1	-1
2	2	2
-1	-1	-1

Figura 2.21: Matriz de pesos

-1	-1	2
-1	2	-1
2	-1	-1

Figura 2.22: Matriz de pesos

2.8 Detecção de arestas ou bordas

Dá-se o nome de **aresta** ou **borda** aos pontos de uma imagem pertencentes à fronteira entre regiões que tenham níveis de cinza diferentes entre si. Do Cálculo Diferencial sabemos que os operadores diferenciais, por exemplo o gradiente, são adequados para detectar mudanças na inclinação de curvas ou superfícies [Apo58].

Dada uma função $f : \mathbb{R}^2 \to \mathbb{R}^2$, o gradiente de f é definido pelo vetor:

$$\nabla f(x, y) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$$

Geometricamente, ele é um vetor que aponta na direção de maior crescimento de f no ponto (x, y) e seu módulo (que é a infomação importante na detecção de arestas) é proporcional à intensidade do crescimento de f.

Para uma imagem digital $f \in K^{\mathbf{E}}$, o módulo do gradiente pode ser aproximado pelo operador [GW92],

$$\mid \nabla f \mid \simeq \mid \frac{\partial f}{\partial x} \mid + \mid \frac{\partial f}{\partial y} \mid \simeq [(w_7+2w_8+w_9)-(w_1+2w_2+w_3)]+[(w_3+2w_6+w_9)-(w_1+2w_4+w_7)]$$

Os termos entre colchetes são denominados **Operadores de Sobel**. Eles podem ser vistos na forma de tabelas na fig. 2.1.

Existem outras formas de aproximar o cálculo do gradiente, mas os operadores de Sobel são os que dão os resultados visualmente mais suaves.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Tabela 2.1: Operadores de Sobel

2.9 Qualidade de uma Segmentação

Um sistema de extração automática de informações deve ter, também, um módulo automático de avaliação da qualidade da imagem segmentada. Encontrar e implementar um bom critério de avaliação do resultado de uma segmentação de imagens não é simples. No entanto, algumas regras já tornaram-se conhecidas e, em geral, são seguidas[Har92, HS85]:

1. Regiões de uma imagem segmentada devem ser uniformes e homogêneas com respeito a algum critério, tal como níveis de cinza ou textura.
2. O interior de uma região segmentada E_i deve ser simples (em geral côncavo) e não deve conter muitos buracos pequenos. As imagens das fig. 2.23 e fig 2.24 ilustram como deve, ou não, ser o interior de uma região segmentada, respectivamente.
3. Regiões adjacentes a uma região segmentada devem ser significativamente diferentes com respeito à característica de uniformidade dos pontos na região segmentada. Por exemplo, se a característica importante é a textura, esta deve ser significativamente diferente de uma região para outra região segmentada.
4. As fronteiras de cada região devem ser finas (em espessura), de preferência não regulares, e precisas em relação aos objetos que estão sendo segmentados.

Neste capítulo vimos um resumo de alguns operadores mais importantes para a segmentação de imagens, que já tornaram-se clássicos.

Nos capítulos seguintes vamos tratar o assunto de segmentação sob a abordagem de uma poderosa teoria algébrica, a Morfologia Matemática.

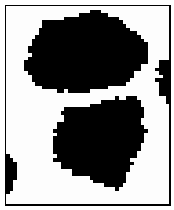

Figura 2.23: Segmentação boa

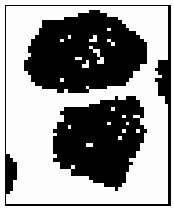

Figura 2.24: Segmentação ruim

Capítulo 3

Elementos de Morfologia Matemática

A Morfologia Matemática tem-se desenvolvido muito nos últimos anos e cada vez mais ela está sendo utilizada por quem trabalha em processamento de imagens.

Neste capítulo faremos uma pequena introdução às principais idéias sob as quais ela está baseada e apresentaremos as operações e operadores mais importantes para a segmentação de imagens.

3.1 Definições Básicas

Definição 3.1.1 *Seja $\mathcal{L}$ um "poset" [Bir67], i.e., um conjunto com uma relação de ordem parcial denotada por $\leq$. $\mathcal{L}$ é um* **reticulado completo** *se e somente se, para toda família $\mathcal{X} \subset \mathcal{L}$, $\mathcal{X}$ possui um ínfimo, denotado por $\bigwedge \mathcal{X}$, e um supremo, denotado por $\bigvee \mathcal{X}$.*

Seja $\leq$ uma relação de ordem entre os elementos de $K^{\mathbf{E}}$, dada por:

$$f \leq g \Longleftrightarrow f(x) \leq g(x), \forall x \in \mathbb{E}$$

É fácil mostrar que ela é uma relação de ordem parcial, i.e., que satisfaz, para todo f, g e h em $K^{\mathbf{E}}$ as propriedades: $f \leq f$; $(f \leq g \text{ e } g \leq h \Longrightarrow f \leq h)$; $(f \leq g \text{ e } g \leq f) \Longrightarrow f = g$.

Teorema 3.1.1 *O conjunto $K^{\mathbf{E}}$ dotado desta relação de ordem é um reticulado completo [BB93].*

No nosso trabalho, restringiremos nosso estudo ao reticulado das funções $(K^{\mathbf{E}})$, que é suficiente para tratar as imagens binárias e as imagens em níveis de cinza.

Definição 3.1.2 *Dado um conjunto $X \subset \mathbb{E}$, o conjunto,*

$$X^t = \{x \in \mathbb{E} : -x \in X\}$$

é o **transposto** *do conjunto* X.

Definição 3.1.3 *Dado um conjunto* $X \subset \mathbb{E}$, *o conjunto,*

$$X^c = \{x \in \mathbb{E} : x \notin X\}$$

é o **complementar** *do conjunto* X.

Seja $X \subset \mathbf{E}$, denotamos por $X+b$, $b \in \mathbf{E}$, a translação de X por b, i.e., $X+b = \{x+b, x \in X\}$.

Seja $f \in K^{\mathbf{E}}$, denotamos por $f+h$, $h \in \mathbf{E}$, a translação de f por h, i.e., $f(x-h)$.

Vamos denotar por $\partial\mathbf{E}$ os pontos limites de $\mathbf{E}$, i.e., a **moldura** de $\mathbf{E}$. Por exemplo, se $\mathbb{E}$ é um subconjunto retangular de $\mathbb{Z} \times \mathbb{Z}$, por exemplo, $[a,b] \times [c,d] \subset \mathbb{Z} \times \mathbb{Z}$, então $\partial\mathbb{E} = \{x \in \mathbb{E} : x = (g,h)$ tal que $g = a,$ ou $g = b$ e $h \in [c,d],$ ou $x = (g,h)$ tal que $h = c,$ ou $h = d$ e $g \in [a,b]\}$.

Definição 3.1.4 *Seja* $f \in \{0,k\}^{\mathbf{E}}$ *tal que,*

$$f(x) = \begin{cases} k & \text{se } x \in \partial\mathbf{E} \\ 0 & \text{caso contrário} \end{cases}$$

é chamada **função moldura** *de* $\mathbb{E}$.

A função constante $f(x) = 1, \forall x \in \mathbf{E}$ será denotada simplesmente por $\mathbb{1}$

Seja $B \subset \mathbb{B}$, $\mathbb{B} = \{-1,0,1\}^2$, i.e., B é um subconjunto do chamado **quadrado elementar**. Dizemos que B é um **losango** ou **cruz** se ele é o quadrado elementar sem os cantos. Dizemos que B é uma **linha** se ele é uma das "hastes" da cruz. Os desenhos nas fig. 3.1 e fig. 3.2 ilustram respectivamente o losango e o quadrado elementar. Os desenhos nas fig. 3.3 e fig. 3.4 ilustram respectivamente a **linha vertical** e a **linha horizontal**.

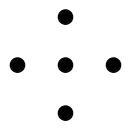

Figura 3.1: Losango

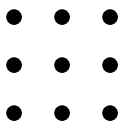

Figura 3.2: Quadrado Elementar

Figura 3.3: Linha Vertical

Figura 3.4: Linha Horizontal

Definição 3.1.5 *Seja d uma métrica [BBL94] sobre $\mathbb{E}$ e r um inteiro positivo ($x \in \mathbb{E}$ e $r \in \mathbb{Z}^+$). O conjunto definido por,*

$$D_d(x,r) = \{y \in \mathbb{E} : d(x,y) \leq r\}$$

é um **Disco Digital**[1] *de centro x e raio r.*

3.2 Máquina Morfológica

A *MM* é uma teoria algébrica que estuda os operadores entre reticulados completos, sejam eles o reticulado Booleano (ou das imagens binárias), ou o reticulado das funções (ou das imagens em níveis de cinza), ou qualquer conjunto que possua as propriedades acima descritas de reticulado completo.

O paradigma central da *MM* sobre funções é a decomposição dos operadores em termos de uma linguagem formal, a **Linguagem Morfológica** ($\mathcal{ML}$), cujo vocabulário são dois operadores (**dilatação** e **erosão**) e três operações (**negação, supremo** e **ínfimo**); os operadores e operações elementares da *MM*.

Esta linguagem é completa, no sentido que ela pode representar qualquer operador entre reticulados, e expressiva, i.e., muitos operadores úteis podem ser representados com frases curtas da $\mathcal{ML}$. Uma frase da $\mathcal{ML}$ é chamada de um **operador morfológico** [BB92, BB91, BB93, BB94].

Uma implementação da $\mathcal{ML}$ é chamada de **Máquina Morfológica** (**MMach**) e um programa de uma MMach é uma implementação de um operador morfológico nesta máquina [BB92]. Neste trabalho restringiremos nossos estudos às MMachs implementadas em máquinas seqüenciais.

3.3 Operadores e Operações elementares

Definição 3.3.1 *Sejam $f, g \in K^{\mathbb{E}}$, as operações, definidas por, para qualquer $x \in \mathbb{E}$,*

$$(f \wedge g)(x) = \min\{f(x), g(x)\}$$

e

$$(f \vee g)(x) = \max\{f(x), g(x)\}$$

são chamadas, respectivamente, **ínfimo** e **supremo** de f e g.

Definição 3.3.2 *Seja $f \in K^{\mathbb{E}}$, a operação definida por, para qualquer $x \in \mathbb{E}$,*

$$\nu(f(x)) = k - f(x)$$

[1] Usaremos simplesmente **disco** em lugar de disco digital daqui em diante.

é chamada de **negação** *de f, onde k é o maior elemento do conjunto K.*

Definição 3.3.3 *Sejam $f, g \in K^{\mathbf{E}}$, o operador definido por, para qualquer $x \in \mathbf{E}$,*

$$(f-g)(x) = \begin{cases} f(x) - g(x) & se\ g(x) < f(x) \\ 0 & caso\ contrário \end{cases}$$

é chamado de **subtração**.

Definição 3.3.4 *Sejam $A, B \in \mathcal{P}(\mathbb{E})$, o operador definido por,*

$$A \oplus B = \cup\{A + b : b \in B\}$$

é a Soma de Minkowski dos conjuntos A e B.

Definição 3.3.5 *Seja $f \in K^{\mathbf{E}}$ e $B \subset \mathbb{E}$, os operadores dados por, para qualquer $x \in \mathbb{E}$,*

$$\delta_B(f)(x) = \bigvee_{y \in B^t + x} f(x+y) = \max\{f(x+y) : y \in B^t + x\},$$

e

$$\varepsilon_B(f)(x) = \bigwedge_{y \in B + x} f(x+y) = \min\{f(x+y) : y \in B + x\}$$

são chamados , respectivamente, **dilatação** e **erosão** de f por B, e o conjunto B é chamado **elemento estruturante**.

As imagens das figuras fig. 3.6 e fig. 3.7 mostram, respectivamente os efeitos desses operadores em uma imagem binária. As imagens das figuras fig. 3.9 e fig. 3.10 mostram, respectivamente, os efeitos desses operadores em uma imagem em níveis de cinza. O elemento estruturante usado é um disco de diâmetro $d = 10$ pontos centrado na origem.

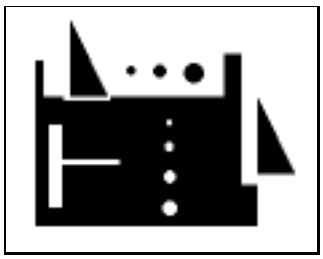

Figura 3.5: Imagem Original

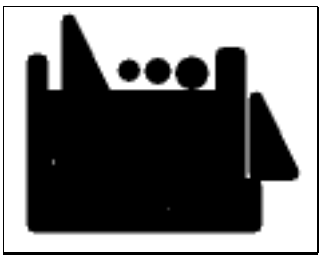

Figura 3.6: Dilatação

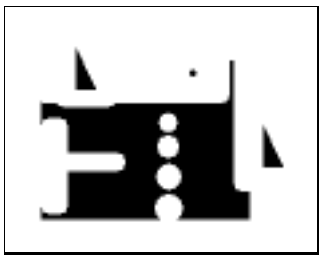

Figura 3.7: Erosão

Observe os seguintes efeitos no resultado da dilatação da imagem binária :

- Os buracos menores do que o elemento estruturante desaparecem.
- Os objetos tornam-se maiores.
- O número de componentes conexas pode diminuir.

Os seguintes efeitos podem ser notados no resultado da erosão binária:

- Os objetos menores do que o elemento estruturante desaparecem.
- Os buracos tornam-se maiores.
- O número de componentes conexas pode aumentar.

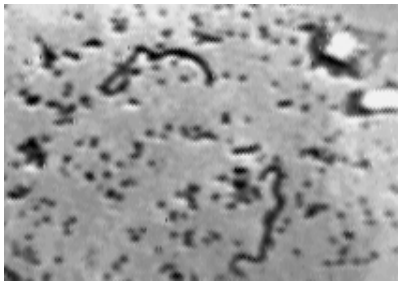

Figura 3.8: Imagem Original

Figura 3.9: Dilatação

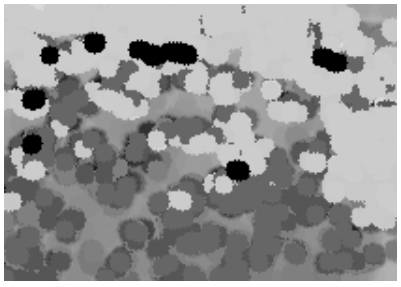

Figura 3.10: Erosão

Para as imagens em níveis de cinza os efeitos são mais suaves em relação às mudanças na forma dos objetos. Note que a quantidade de partes claras da imagem resultante da dilatação (fig. 3.9) é maior do que na imagem original (fig. 3.8). Por outro lado, a quantidade de partes escuras diminui em relação à imagem original. No resultado da erosão (fig. 3.10) acontece o contrário, a quantidade de partes escuras aumenta, diminuindo a quantidade de partes claras.

A noção de elemento estruturante pode ser generalizada pela próxima definição.

Definição 3.3.6 Função estruturante *é uma função que associa a cada ponto p de* $\mathbb{E}$ *um elemento estruturante* $B \subset \mathbb{E}$.

$$b : \mathbf{E} \to P(\mathbf{E})$$

A dilatação ou erosão por uma função estruturante é útil para a construção de operadores que transformam a imagem diferentemente de acordo com a posição do ponto que está sendo transformado, como veremos no capítulo 7.

A partir dessas pequenas observações do comportamento dos operadores elementares, podemos construir operadores úteis para a segmentação de imagens compondo-os de maneira adequada. Nas definições seguintes vamos mostrar alguns exemplos dessas composições.

Definição 3.3.7 *O operador definido por,*

$$\imath(f) = f$$

é chamado **Operador Identidade**.

Este operador é, ao mesmo tempo, uma erosão e uma dilatação por $B = \{\mathcal{O}\}$, onde $\mathcal{O}$ é a origem de $\mathbb{E}$.

Definição 3.3.8 *Seja* $f \in K^{\mathbf{E}}$, $B \subset \mathbf{E}$ *e* $n \in \mathbb{Z}, n > 0$, *os operadores definidos por,*

$$\delta_B^n(f) = (\delta_B(f))^n = \underbrace{\delta_B(f) \circ \delta_B(f) \circ \ldots \circ \delta_B(f)}_{n \text{ operadores}}$$

e

$$\varepsilon_B^n(f) = (\varepsilon_B(f))^n = \underbrace{\varepsilon_B(f) \circ \varepsilon_B(f) \circ \ldots \circ \varepsilon_B(f)}_{n \text{ operadores}}$$

são, respectivamente, a n-**dilatação** e a n-**erosão**.

Definição 3.3.9 *Seja* $f, g \in K^{\mathbf{E}}$ *e* $B \subset \mathbf{E}$, *os operadores definidos por,*

$$\delta_{B,g}(f) = \delta_B(f) \wedge g$$

e

$$\epsilon_{B,g}(f) = \varepsilon_B(f) \vee g$$

são, respectivamente, a **dilatação condicional** *e a* **erosão condicional**.

Note que, pela definição de dilatação condicional, se $f > g$ e $\mathcal{O} \in B$, então $\delta_{B,g}(f) = g$, pois $\delta_B(f) \geq f$. Assim, para a operação fazer sentido nesse caso, a condição $f \leq g$ deve valer. O contrário também é válido para o caso da erosão condicional.

No caso dos operadores condicionais, é usual chamar a função f de **imagem marcadora** e a função g de **imagem máscara**.

Definição 3.3.10 *Seja $f, g \in K^{\mathbb{E}}$, $B \subset \mathbb{E}$ e $n \in \mathbb{Z}, n > 0$, os operadores definidos por,*

$$\delta^n_{B,g}(f) = (\delta_{B,g}(f))^n = \underbrace{\delta_{B,g}(f) \circ \delta_{B,g}(f) \circ \ldots \circ \delta_{B,g}(f)}_{n \text{ operadores}}$$

e

$$\varepsilon^n_{B,g}(f) = (\varepsilon_{B,g}(f))^n = \underbrace{\varepsilon_{B,g}(f) \circ \varepsilon_{B,g}(f) \circ \ldots \circ \varepsilon_{B,g}(f)}_{n \text{ operadores}}$$

*são, respectivamente, a n-**dilatação condicional** e a n-**erosão condicional**.*

A n-composição dos operadores condicionais aproximam a função marcadora da imagem máscara tanto quanto queiramos, a n-dilatação condicional aproxima a função por baixo (no sentido que ela sempre será menor do que a imagem máscara) e a n-erosão condicional aproxima a função por cima.

A composição de dilatações ou erosões condicionais até a estabilidade, i.e., até que nenhum ponto da imagem composta tenha o seu valor modificado, forma operadores importantes para a segmentação.

Definição 3.3.11 *Seja $f, g \in K^{\mathbb{E}}$ e $B \subset \mathbb{E}$, os operadores formados pela iteração até a estabilidade da dilatação ou erosão condicionais definidos por,*

$$\rho_{B,g}(f) = \bigvee_{n \geq 1} \delta^n_{B,g}(f).$$

e

$$\rho^*_{B,g}(f) = \bigwedge_{n \geq 1} \varepsilon^n_{B,g}(f).$$

são, respectivamente, a **recostrução** *e a* **reconstrução dual**.

Os operadores de reconstrução são muito úteis em segmentação pois é comum querermos reconstruir os objetos de interesse de uma imagem a partir de marcadores para esses objetos.

Definição 3.3.12 *Seja $f \in K^{\mathbb{E}}$ e $p \in \mathbb{E}$, dizemos que p é um* **máximo local** *da função f se $f(p) > f(q), \quad \forall q \in N_G(p)$.*

O operador reconstrução pode ser usado para achar os máximos locais de uma função e diversos outros operadores úteis para a segmentação de imagens.

Definição 3.3.13 *Seja $f \in K^{\mathbf{E}}$, o operador definido por,*

$$Max(f) = \rho_{B,f}(f - \mathbb{1}),$$

é uma função binária cujos pontos p tais que $Max(p) \neq 0$, correspondem aos máximos locais da função f.

Definição 3.3.14 *Seja $f \in K^{\mathbf{E}}$, o operador definido por,*

$$T = \imath - \rho_{B,f}(\epsilon_B)$$

pertence à classe dos operadores "top-hat" (que são os operadores definidos pelo resíduo de uma identidade por uma abertura [Ser82, Ser88]).

Definição 3.3.15 *Seja $f \in \{0, l\}^{\mathbf{E}}$, $l \in K$, $l \leq k$, o operador definido por,*

$$\Lambda(f)(x) = \begin{cases} i & se\ \exists i \in [1, n]\ tal\ que\ x \in X_i \\ 0 & caso\ contrário \end{cases},$$

onde $X_1, X_2, \ldots, X_n$ são as componentes conexas de f, é chamado **Rotulação**.

A erosão e a dilatação pelo mesmo elemento estruturante podem ser compostas para formar dois novos operadores. Quando a erosão é aplicada após uma dilatação o operador é chamado *fechamento*. Quando a dilatação é aplicada após a erosão, o operador é chamado *abertura*. Estes operadores são filtros morfológicos[2].

Definição 3.3.16 *Seja $f \in K^{\mathbf{E}}$ e $B \subset \mathbf{E}$, os operadores definidos por,*

$$\gamma_B(f) = \delta_B(\epsilon_B)(f)$$

e

$$\phi_B(f) = \epsilon_B(\delta_B)(f)$$

são chamados, respectivamente, de **abertura** *e* **fechamento** *de f.*

Definição 3.3.17 *O operador definido por,*

$$\varrho_B = \bigvee_{i=0,1,\ldots} (\epsilon_B^i - \rho_{B,\epsilon_B^{i+1}\epsilon_B^i})$$

é chamado de **erosão última** *de parâmetro B.*

[2]Um operador morfológico Ψ é dito ser um filtro se ele for crescente, i.e., se dados $f, g \in K^{\mathbf{E}}, f \leq g$, então $\Psi(f) \leq \Psi(g)$, e idempotente, i.e., $\Psi(\Psi(f)) = \Psi(f)$ [SS95, Ser82, Ser88, Ser94, SV92, Hei95, Hei91].

Definição 3.3.18 *Seja $f \in \{0,k\}^{\mathbf{E}}$, o operador definido por,*

$$\Phi_B(g) = \nu(\rho_{B,f})(\nu(g))$$

é chamado de **Fechamento de Buracos**.

O operador que apresentaremos em seguida serve para eliminar os objetos que tocam a moldura de $\mathbb{E}$.

Definição 3.3.19 *Seja $f \in \{0,k\}^{\mathbf{E}}$, o operador definido por,*

$$\Gamma_B(g) = \rho_{B,f}(g)$$

é chamado de **"Edge Off"**.

Definição 3.3.20 *Seja $f \in \mathbb{E}$ e $B \subset \mathbf{E}$, B igual ao losango ou ao quadrado elementar, o operador definido por,*

$$\Psi_B(f) = \delta_B(f) - \epsilon_B(f)$$

é o **Gradiente Morfológico** *da função f.*

Definição 3.3.21 *Seja $f \in K^{\mathbf{E}}$ e $B \subset \mathbf{E}$ os operadores definidos por,*

$$\delta_B^a(f) = \nu(\delta_{B^c}(f))$$

e

$$\varepsilon_B^a(f) = \nu(\varepsilon_B(f))$$

são, respectivamente, a **anti-dilatação** *e a* **anti-erosão**.

Definição 3.3.22 *Sejam $A, B \subset \mathbf{E}$ tal que $A \subset B$, os operadores definidos por,*

$$\lambda_{A,B} = \epsilon_A \wedge \delta_{B^t}^a$$

e

$$\mu_{A,B} = \delta_A \wedge \epsilon_{B^{ct}}^a$$

são, respectivamente, a **sup-geradora** *e a* **inf-geradora** *de parâmetros A e B.*

Definição 3.3.23 *Sejam $A, B^c \subset \mathbf{E}$ tal que $A \subset B$, os operadores definidos por,*

$$\sigma_{A,B} = i \wedge \lambda_{A,B}$$

e

$$\tau_{A,B} = i \vee \lambda_{A,B}$$

são, respectivamente, o **Afinamento** *e o* **Espessamento** *de parâmetros A e B.*

Definição 3.3.24 *Sejam A e $B^c \subset \mathbf{E}$ tais que $A \subset B$. Seja $g \in K^{\mathbf{E}}$, os operadores definidos por,*

$$\sigma_{A,B,g} = \sigma_{A,B} \vee g$$

e

$$\tau_{A,B,g} = \tau_{A,B} \wedge g$$

são chamados, respectivamente, **Afinamento condicional** *e* **Espessamento condicional.**

Tanto o afinamento (espessamento) quanto o afinamento condicional (espessamento condicional) podem ser compostos com eles mesmos para formar os n-operadores.

Definição 3.3.25 *Sejam $\mathcal{A}$ e $\mathcal{B}$ duas seqüências infinitas de elementos estruturantes primitivos, respectivamente, com elementos A_i e B_i tais que $A_i \subset B_i$. Os operadors definidos por,*

$$\Sigma_{\mathcal{A},\mathcal{B}} = \sigma_{A_1,B_1} \ldots \sigma_{A_i,B_i} \ldots$$

$$\Theta_{\mathcal{A},\mathcal{B}} = \tau_{A_1,B_1} \ldots \tau_{A_i,B_i} \ldots$$

são chamados, respectivamente, de **Esqueleto por Afinamento** *e* **Exoesqueleto por Espessamento** *de parâmetros $\mathcal{A}$ e $\mathcal{B}$.*

Seja $A \subset \mathbf{E}$ e $A_{i\alpha}$ um conjunto formado pela rotação no sentido horário de $i\alpha$ graus de A. Seja $\mathcal{A}_\alpha$ uma seqüência infinita com elementos $A_i = A_{i\alpha}$.

Por exemplo, seja $A \subset \mathbf{E}$ definido pela seguinte matriz,

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

então $\mathcal{A}_{45}$ será a seguinte seqüência,

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \cdots$$

Sejam $A, B \subset \mathbf{E}$ definidos pelas seguintes matrizes, respectivamente,

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

O esqueleto por afinamento de parâmetros $\mathcal{A}_{45}$ e $\mathcal{B}_{45}$, $\Sigma_{\mathcal{A}_{45},\mathcal{B}_{45}}$, será chamado de **Esqueleto Homotópico**.

Definição 3.3.26 *Seja $B \subset \mathbb{E}$, o operador definido por,*

$$\sigma_B = \bigvee_{i=1,2,\ldots} (\epsilon_B^i - \rho_B \epsilon_B^i)$$

é chamado de **Esqueleto Morfológico** *de parâmetro B.*

Definição 3.3.27 *Seja d uma métrica sobre $\mathbb{E}$ e $f \in K^{\mathbf{E}}$. O operador definido por,*

$$\Psi_d(f)(x) = min\{d(x,y) : y \in \mathbb{E} e f(y) = 0\}$$

é chamado de **Função Distância**.

É possível provar que as funções distância são erosões [BBL94].

Definição 3.3.28 *Seja $A \subset \mathbb{E}$ e d uma métrica sobre $\mathbf{E}$, a função definida por,*

$$d_A(x,y) = min\{l(P) : P \text{é um caminho entre } x \text{ e } y \text{ totalmente contido no conjunto } A\}$$

é a **distância geodésica** *[Ros78, BM93] entre x e y.*

A noção de distância geodésica é útil para simplificar a definição de certos operadores morfológicos. O desenho da fig. 3.11 ilustra este conceito.

Seja $A \subset \mathbb{E}$, $x, y \in A$, $B \subset A$, $x \notin B$ e $d_A(x,y)$ a distância geodésica de x a y em A. A função definida por,

$$d_A(x,B) = \min\{d_A(x,y) : y \in B\}$$

é a distância geodésica do ponto x ao conjunto $B \subset A$ [VS91, BM93].

Definição 3.3.29 *Seja B um conjunto formado por $B_1, B_2, \ldots, B_k$, k componentes conexas de $A, A \subset \mathbf{E}$. O conjunto definido por,*

$$i_z a(B_i) = \{p \in A, \forall j \in [1,k]/\{i\}, d_A(p,B_i) < d_A(p,B_j)\}$$

é chamado de **zona de influência geodésica** *da componente B_i.*

O desenho da fig. 3.12 ilustra esse conceito. Os pontos que não pertencem a nenhuma zona de influência geodésica pertencem ao esqueleto por zonas de influência ("Skeleton by Influence Zones") ou **SKIZ** de B dentro de A e é denotado $SKIZ_A(B)$.

Definição 3.3.30 *O conjunto definido por,*

$$SKIZ_A(B) = A/IZ_A(B),$$

onde $IZ_A(B) = \bigcup_{i \in [1,k]} iz_A(B)$, é chamado de **SKIZ** *de B.*

Uma definição de SKIZ que exprime melhor o operador em termos de operadores elementares é dada abaixo.

Sejam $C, D \subset \mathbf{E}$ definidos pelas seguintes matrizes, respectivamente,

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Definição 3.3.31 *Seja $f \in \{0,k\}^E$, A e B as mesmas matrizes usadas para o esqueleto homotópico, C e D as matrizes definidas acima e $\mathcal{A}_{45}$, $\mathcal{B}_{45}$, $\mathcal{C}_{45}$ e $\mathcal{D}_{45}$ seqüências infinitas de elementos estruturantes primitivos definidos por A, B, C e D. O operador definido por,*

$$\Sigma(f) = \Sigma_{\mathcal{C}_{45},\mathcal{D}_{45}}(\Sigma_{\mathcal{A}_{45},\mathcal{B}_{45}}(\nu(f)))$$

é chamado SKIZ de f.

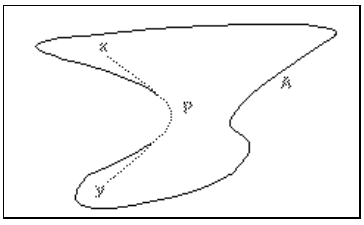

Figura 3.11: Distância Geodésica

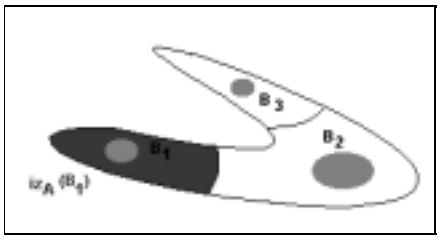

Figura 3.12: Zona de influência geodésica

3.4 Linhas de Partição de Águas

No contexto da Morfologia Matemática, um dos operadores que tem-se mostrado muito útil e eficiente para a segmentação de imagens, mais especificamente a identificação de contornos de objetos numa imagem em níveis de cinza, é a Linha de Partição de Águas (**LPE**) [BM93, Vin90, VS91] introduzido por Beucher e Meyer.

A idéia deste operador pode ser entendida intuitivamente se pensarmos que uma imagem em níveis de cinza pode ser vista como um relevo onde encontramos partes que se assemelham a vales e outras a montanhas. A maioria das pessoas já vivenciou alguma vez, o processo de formação de poças d'água em uma superfície rugosa quando está chovendo. Ao observamos com cuidado esse processo, vemos que a medida que a água vai enchendo os vales e formando poças, elas vão crescendo e juntando-se a outras poças, formadas pela acumulação d'água em vales vizinhos. O nome que se dá aos vales onde acumulam-se as poças é **"Bacias de Captação"**. Ao lugar

geométrico formado pelos pontos onde ocorre a junção das poças, dá-se o nome de **"Linhas de Partição de Águas"**.

Vamos formalizar estes conceitos, em seguida, apresentando uma definição do operador baseada em "imersão"; que é a base para o algoritmo de LPE que estudamos, implementamos e que apresentaremos no capítulo 6. Uma definição baseada em operadores elementares pode ser encontrada em [BBL94].

Definição 3.4.1 *Seja $f \in K^{\mathbf{E}}$, dizemos que $R_h \subset \mathbb{E}$ é um* **Mínimo Regional** *de altitude h se R_h é um conjunto conexo de pontos de $\mathbb{E}$ tal que $f(p) = h, \forall p \in R_h, R_h \subset \mathbb{E}$.*

Chamamos de $RM_h(f)$ o conjunto dos mínimos regionais de f de altitude h.

Definição 3.4.2 *Seja h_{min} e h_{max} o menor e o maior valor, respectivamente, de $f \in K^{\mathbf{E}}$, i.e., $h_{min} = inf(f(x))$ e $h_{max} = sup(f(x))$.*

Definição 3.4.3 *Seja $f \in K^{\mathbf{E}}$, o conjunto definido por,*

$$f^h = \{x \in \mathbb{E} : f(x) \leq h\}$$

é chamado de seção inferior de f no nível h.

Definição 3.4.4 *O conjunto $X_{h_{max}}$ obtido pela relação de recorrência,*

$$X_{h_{min}} = f^{h_{min}}(f)$$

$$X_{h+1} = RM_{h+1}(f) \cup IZ_{f^{h+1}}(X_h)$$

para todo $h \in [h_{min}, h_{max} - 1]$, é o conjunto das **Bacias de Captação** *de f.*

O desenho da fig. 3.13 mostra a dinâmica da relação de recorrência acima.

Definição 3.4.5 *Seja $f \in K^{\mathbf{E}}$, o conjunto complementar das Bacias de Captação de f são as* **Linhas de Partição d'Águas** *da função f.*

A implementação mais eficiente deste operador LPE foi feita por Vincent e Soile [Vin90, VS91]. O algoritmo implementado por eles usa a idéia acima para encontrar as linhas de partição da seguinte maneira: no ponto mais baixo de cada vale (ou bacia de captação) imaginemos que há um furo. Em seguida, simulemos a formação das poças através da imersão dessa superfície em um tanque d'água de modo que a superfície seja inundada através daqueles furos.

Conforme as bacias de captação vão enchendo e se juntando, marca-se os pontos de encontro de uma bacia com a outra. O conjunto desses pontos marcados formam as linhas de partição d'águas procuradas.

Este algoritmo, juntamente com outros algoritmo rápidos, serão apresentados e analisados no capítulo 6.

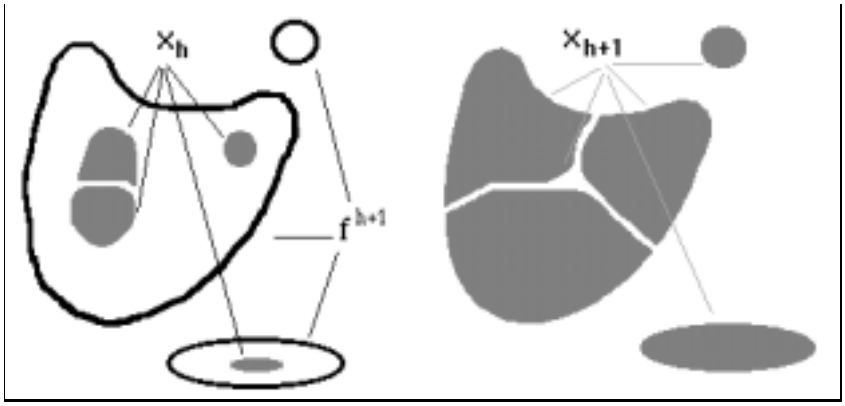

Figura 3.13: Dinâmica da recorrência

Capítulo 4

Segmentação Morfológica

Neste capítulo apresentaremos algumas idéias de como operadores entre imagens, principalmente os operadores para segmentação, podem ser construídos de maneira "ad-hoc" utilizando-se composições dos operadores morfológicos básicos apresentados anteriormente. Em seguida, veremos um modelo de segmentação de imagens que utiliza o operador de linha de partição de águas (LPE) e que tem-se mostrado muito útil em aplicações reais complexas. Finalmente, mostraremos a equivalência entre a segmentação através dos operadores clássicos apresentados no capítulo 2 e a Segmentação Morfológica.

No próximo capítulo daremos alguns exemplos práticos de como projetar operadores para resolver os problemas de extração de informações de uma imagem e, também, exemplos do modelo baseado no operador LPE.

4.1 Projeto "ad-hoc" de operadores

Vimos que a Morfologia Matemática é uma teoria útil para extrair informação de imagens e, sob este enfoque, os problemas de visão computacional são resolvidos por transformações entre imagens que são os operadores entre reticulados.

Estes operadores são projetados como composições de operadores e operações elementares da teoria e expressam o conhecimento do usuário a respeito de um problema específico. No momento não há regras para a escolha correta destes operadores, assim, o projeto de operadores é uma arte cuja destreza adquire-se apenas com muita experiência e conhecimento do assunto.

A idéia principal de um processo "ad-hoc" de projeto de operadores é a decomposição do objetivo final em vários subobjetivos, para expressá-los como operadores específicos entre reticulados. Desta maneira, o objetivo final será a composição dos operadores entre reticulados que representam cada um dos subobjetivos.

Vamos dar um exemplo de caráter ilustrativo que mostra a classificação de objetos de duas dimensões pelos seus comprimentos. Na prática, sistemas automáticos de classificação de objetos por tamanho são úteis em diversos lugares, principalmente em indústrias. O tamanho de um

objeto é proporcional ao número de pontos de sua representação na imagem assim, se conseguirmos segmentar o objeto na imagem, podemos calcular e trabalhar com suas dimensões visíveis na imagem, facilmente.

Nosso objetivo será construir um operador para segmentar os objetos maiores que um certo comprimento dado da imagem da fig. 4.1, utilizando os operadores básicos da *MM*.

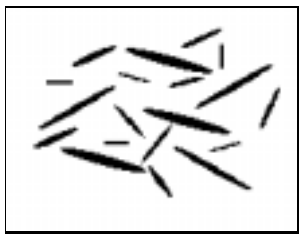

Figura 4.1: Elipses de diversos tamanhos

Primeiramente, vamos encontrar um operador que aplicado à imagem da fig. 4.1 resulte numa outra que contenha um modelo simplificado dos objetos e que ainda contenha a informação que precisamos para separá-los, no caso o comprimento.

O operador da *MM* que faz isso é o *esqueleto homotópico* [BB94]. O resultado é mostrado na fig. 4.2.

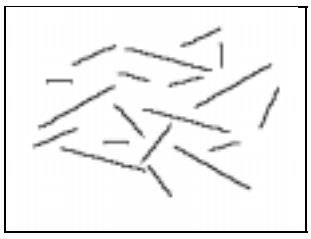

Figura 4.2: Esqueleto da imagem

Em seguida, precisamos de um operador que separe os objetos mais longos da fig. 4.2, dos outros. Para isso, aplicamos um *emagrecimento controlado* [BB94] sobre a imagem resultante do esqueleto morfológico, diminuindo o tamanho dos objetos a partir dos extremos; o resultado (fig. 4.3), será apenas o que resta dos objetos maiores que um certo comprimento desejado.

A estes objetos resultantes do emagrecimento, damos o nome de marcadores. Em geral, marcadores são subconjuntos dos objetos que desejamos segmentar e o valor atribuído aos pontos desses subconjuntos, neste caso, é igual ao valor deles na imagem original. Eles têm esse nome pois "marcam" os objetos a serem segmentados.

Para finalizar o problema, aplicamos um operador que restaure os objetos da fig. 4.1 para o seu tamanho original. O operador de reconstrução morfológica reconstrói os objetos de maior

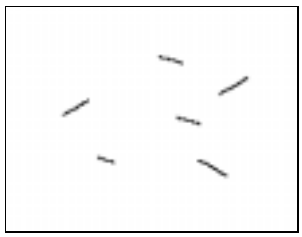

Figura 4.3: Marcadores para as elipses de maior comprimento

comprimento da imagem original a partir dos marcadores da imagem da fig. 4.3. O resultado final pode ser visto na imagem da fig. 4.4.

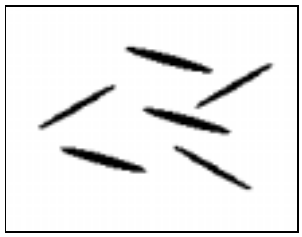

Figura 4.4: Resultado final da segmentação

Para realizar esta segmentação nós nos utilizamos de três operadores importantes da *MM*, quais sejam: o esqueleto morfológico, o emagrecimento controlado pelos extremos e a reconstrução morfológica. A composição destes operadores resulta num operador que separa os objetos de comprimento maior que um certo valor dado. É importante notar que este operador pode não funcionar corretamente para imagens cujos objetos não tenham características semelhantes às da imagem para a qual ele foi construido.

Vamos reforçar a importância de dividirmos o objetivo principal de um problema em subobjetivos mais simples, examinando um pouco mais o exemplo anterior.

Inicialmente não tínhamos um operador que separasse objetos de formato qualquer pelos seus comprimentos. A segmentação foi possível pois temos o operador de emagrecimento pelos extremos, que pode ser usado para separar objetos maiores que um certo comprimento, desde que eles tenham a espessura de um ponto. Por isso é que foi necessário a aplicação de um operador que transformasse os objetos para ficarem com a espessura de um ponto, sem perder a informação do comprimento. Este operador (emagrecimento pelos extremos), por sua vez, também foi construído a partir da solução de objetivos mais simples, i.e., a partir de um operador que elimina pontos estremos, e assim por diante. Este processo de simplificação ocorre até o momento em que só temos objetivos que podem ser realizados pelos operadores elementares da *MM*. Como sabemos que a combinação correta dos operadores elementares pode resultar em qualquer operador entre imagens [BB93], se nossa heurística for adequada, podemos construir operadores para resolver

qualquer problema de PDI, no nosso caso, qualquer problema de segmentação de imagens.

4.2 Paradigma de Beucher

Já vimos que os métodos existentes para segmentar uma imagem utilizam basicamente duas idéias. Uma delas é achar os contornos dos objetos na imagem. A outra, é agrupar pontos que tenham **características** semelhantes até que o objeto de interesse seja reconstruído. O Paradigma de Beucher é um método baseado na extração dos contornos dos objetos a serem segmentados. O problema de identificação de contornos pode ser resolvido utilizando-se o operador LPE, visto no capítulo 3, em conjunto com outros operadores morfológicos que servem para "preparar" a imagem. Essa preparação deve ocorrer para o realce e identificação dos objetos que devem ser segmentados.

Contornos, em geral, são descontinuidades na imagem e estas podem ser realçadas através dos operadores de diferenciação. Normalmente, usa-se o operador gradiente morfológico para isso.

No capítulo 3, vimos que a aplicação do gradiente morfológico em uma imagem em níveis de cinza resulta em uma outra imagem em níveis de cinza com picos e vales onde estão as descontinuidades; tanto maior será a altura dos picos quanto maior forem as descontinuidades. Na imagem da fig. 4.6 mostramos o inverso do resultado do operador gradiente morfológico aplicado à imagem da fig. 4.5, donde queremos segmentar as células escuras [1].

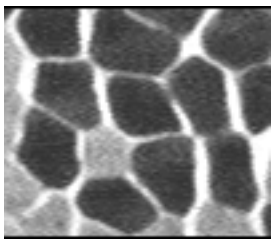

Figura 4.5: Músculo

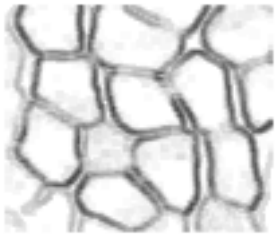

Figura 4.6: Gradiente invertido

É importante notar que o operador gradiente é muito sensível a ruídos, i.e., a variações de intensidade nos níveis de cinza da imagem, daí a grande quantidade de picos e vales na imagem (partes escuras e claras).

Um problema muito conhecido [BM93, Vin90, VS91] é a aplicação do operador LPE sobre a imagem do gradiente que, em geral, resulta numa super-segmentação da imagem (maior número de linhas de partição do que de objetos na imagem, fig. 4.7). Isto é de se esperar já que cada vale na imagem é uma bacia de captação d'águas para o operador LPE e porque a imagem resultante da aplicação do operador gradiente morfológico pode possuir muitos vales devido à sua sensibilidade a ruídos.

[1] É fácil ver que a aplicação um simples threshold nessa imagem pode separar as células escuras do fundo da imagem, mas não as separa entre si. Daí termos de usar outras ferramentas de segmentação.

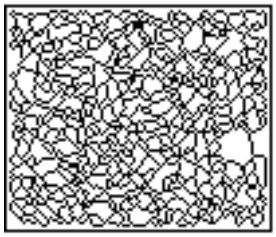

Figura 4.7: Supersegmentação

Figura 4.8: Marcadores

Filtrar a imagem original nem sempre é uma boa solução para forçar o desaparecimento dos ruídos pois pode acarretar o desaparecimento de contornos importantes[2]. A filtragem posterior à aplicação do operador LPE sobre o resultado do gradiente, com intuito de diminuir a super-segmentação, também pode acarretar a perda ou o desaparecimento de contornos, como exemplificado em [BM93].

A técnica mais eficiente para resolver o problema acima é a Mudança de Homotopia do Gradiente, que consiste em, antes da aplicação do operador LPE, filtrar a imagem resultante da aplicação do gradiente para eliminar as descontinuidades responsáveis pela super-segmentação. Esta técnica é devida a F. Meyer e está descrita, juntamente com outras técnicas para resolver o problema da super-segmentação, em [BM93].

Mudança de Homotopia do Gradiente

Para fazer a mudança de homotopia do gradiente é preciso introduzir uma nova fase ao processo de segmentação que depende do conhecimento prévio dos objetos a serem segmentados, i.e., temos que introduzir a informação de quais bacias são importantes e quais não, baseado nas características dos objetos que queremos segmentar. Essa informação é introduzida através da escolha (automática, ou não) de regiões conexas $\{\mathbb{E}_\lambda\}$, $\mathbb{E}_i \subset \mathbb{E}$, de pontos da imagem $\mathbb{E}$, que de agora em diante chamaremos de **marcadores**. A finalidade de um marcador será indicar qual objeto deve ser segmentado. Por exemplo, a imagem da figura 4.8 contém os marcadores para as células escuras da imagem da fig. 4.5.

Definição 4.2.1 *Seja $f \in K^{\mathbb{E}}$, o resultado da aplicação do operador gradiente a uma imagem $h \in K^{\mathbb{E}}$. Seja $\{\mathbb{E}_\lambda\}$, $\mathbb{E}_i \subset \mathbb{E}$, uma família de marcadores para os objetos a serem segmentados na imagem h. Seja $g \in \mathbb{E}^K$ a função formada atribuindo-se o valor k_M, onde k_M é um majorante de h, aos pontos pertencentes aos marcadores ($x \in \mathbb{E}_i$) e k_m, onde k_m é um minorante de h, caso contrário. O operador de mudança de homotopia do gradiente é definido por:*

$$\theta(f) = \rho^*_{(f \wedge \nu(g))}(\nu(g)),$$

[2]Neste problema fizemos uma pequena filtragem como parte da segmentação das células claras, mas que para as células escuras não afeta o resultado. A solução completa do problema será apresentada no próximo capítulo.

onde ρ^* é o operador de reconstrução dual.

Vamos exemplificar como são usados os marcadores para modificar o resultado do gradiente morfológico, usando uma imagem de uma dimensão para ajudar a visualização da dinâmica do processo.

As imagens da fig. 4.9 e da fig. 4.10 mostram uma função h e a função f resultante da aplicação do gradiente morfológico em h, respectivamente. Nosso objetivo será segmentar o objeto representado pelos valores entre 2 e 22 na imagem da fig. 4.9. Note que há um vale importante correspondente a essa região na imagem da fig. 4.10 e vários outros vales devidos a ruídos na imagem. A aplicação do operador LPE sobre a imagem da fig. 4.10 resultaria numa imagem com excesso de contornos, como mencionado acima, pois cada vale corresponde a uma bacia de captação. A função g, mostrada na fig. 4.11, foi obtida por uma limiarização da função h que segmentou apenas o ponto mais baixo do vale. Ela será a imagem do marcador para o objeto que queremos segmentar e a imagem da fig. 4.12 mostra o resultado do ínfimo entre as funções f e $\nu(g)$.

As imagens mostradas em fig. 4.13, fig. 4.14, fig. 4.15 e fig. 4.16, mostram algumas etapas do processo de mudança de homotopia da imagem da fig. 4.10. Nestas imagens pode-se ver a função $\nu(g)$ e o resultado do ínfimo entre f e $\nu(g)$. O resultado final é mostrado na fig. 4.17.

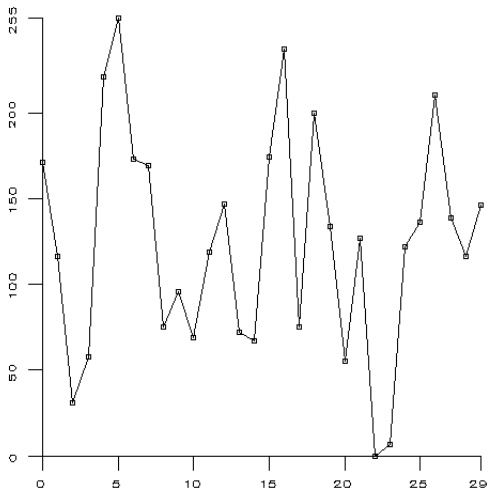

Figura 4.9: Função unidimensional h

O operador $\theta(f)$ é um filtro [SV92] que muda apenas os mínimos regionais de f (não há alteração dos máximos regionais). É deste fato que pode-se dizer que $\theta(f)$ muda a homotopia de f pois ele muda a homotopia baixa de f [Mey91]. Isto é importante pois implica que as linhas encontradas após a aplicação da mudança de homotopia são um subconjunto das linhas de partição da imagem do gradiente, i.e., não há acréscimo de picos (máximos regionais), apenas diminuição do número de vales (mínimos regionais).

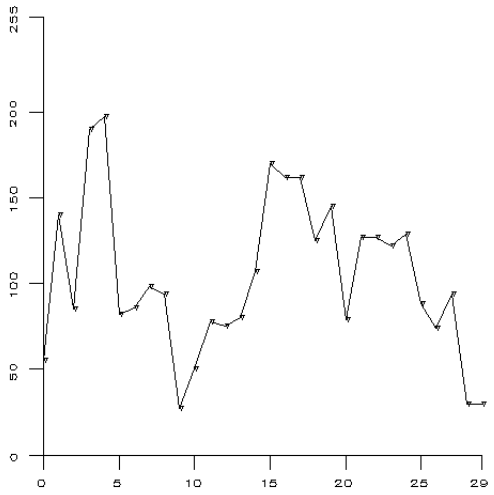

Figura 4.10: Gradiente de h

Após a eliminação dos mínimos desnecessários, é fácil ver que a aplicação do operador LPE produzirá o resultado desejado. No caso do problema das células escuras, a filtragem (fig. 4.18) conduz a um resultado satisfatório após a aplicação do operador LPE, como pode ser visto na imagem da fig. 4.19.

O modelo de segmentação de Beucher [BM93, Beu90, MB90, SV92] pode ser resumido no fluxograma mostrado na fig. 4.20.

No lado direito da figura temos a parte heurística do método, que é o projeto de operadores para achar marcadores na imagem para os objetos de interesse.

Na parte esquerda do fluxograma temos as etapas anteriores à aplicação da mudança de homotopia do gradiente da imagem. Nesta etapa, após a observação e análise visual da imagem, pode-se aplicar filtragens para que o resultado do gradiente seja o melhor possível. Deve-se tomar muito cuidado nisto para evitar o desaparecimento de contornos importantes.

Resulta destas etapas a imagem dos marcadores e a imagem do gradiente. O passo seguinte é fazer a negação da imagem dos marcadores e o ínfimo deste resultado com a imagem original. O resultado do ínfimo será usado para fazer a mudança da homotopia do gradiente. O resultado deste operador será uma imagem com os contornos dos objetos de interesse realçados e sem as descontinuidades devidas aos ruídos. É como se passássemos um trator em cima da superfície, tirando as pequenas colinas e enchendo os pequenos vales, fazendo com que tenhamos apenas uma bacia de captação para cada marcador construído.

A aplicação do operador LPE produzirá, então, uma imagem com as linhas de partição bem posicionadas sobre os contornos dos objetos.

A marcação pode ser feita manualmente para algumas imagens, entretanto é comum estarmos interessados em obter uma metodologia robusta para a geração de marcadores, para que ela sirva

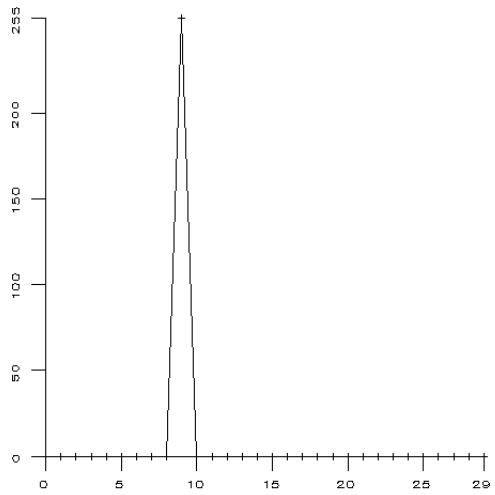

Figura 4.11: Função Marcadora

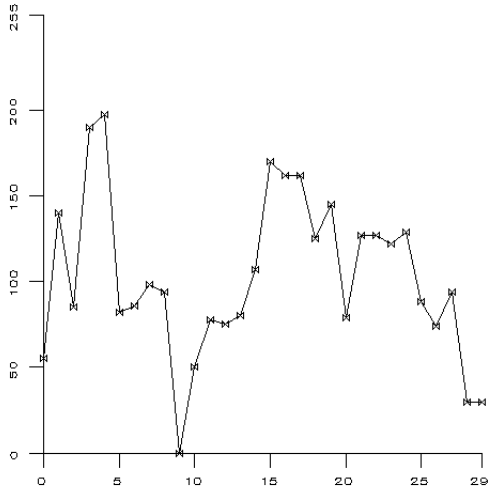

Figura 4.12: Ínfimo

para outras imagens com características semelhantes.

Através do paradigma de Beucher, o problema de deteção de contornos que, em geral, é muito complicado, é substituido por um problema mais simples, que é o de achar marcadores para os objetos de interesse na imagem.

Atualmente, os operadores para produzir os marcadores são projetados de forma "ad-hoc". É difícil evitar esta parte heurística no processo de segmentação já que a segmentação envolve um conhecimento a priori dos objetos a serem segmentados. Por exemplo, um médico acostumado a examinar o resultado de exames de R-X pode facilmente ver um problema como um cisto na imagem, enquanto um leigo teria dificuldade para interpretar os objetos na imagem. Neste caso, o médico ensinaria o profissional em processamento de imagens para fazer um operador que marcasse e segmentasse cistos na imagem.

No futuro, com o aperfeiçoamento dos processos de projeto automático de operadores, como os baseados no aprendizado PAC [BdSB94, BTdST95, BTdST96], por exemplo, estas tarefas poderão ser automatizadas.

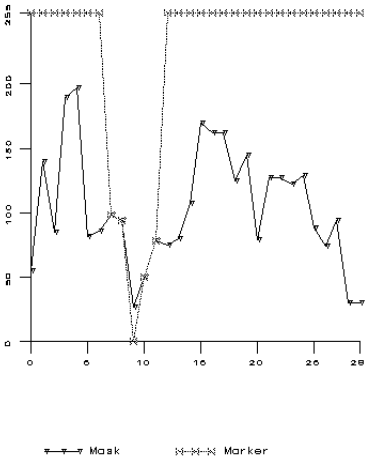

Figura 4.13: Resultado após duas iterações

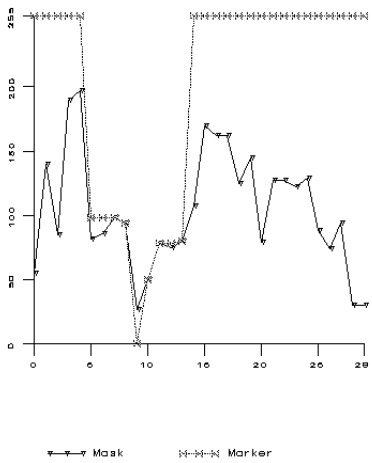

Figura 4.14: Resultado após quatro iterações

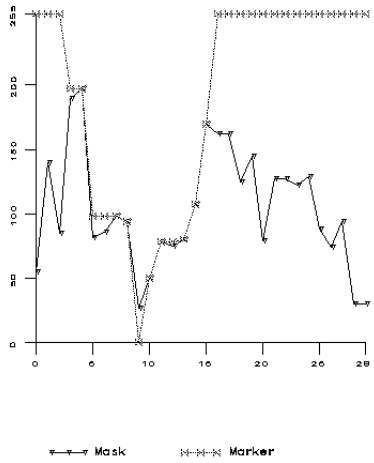

Figura 4.15: Resultado após seis iterações

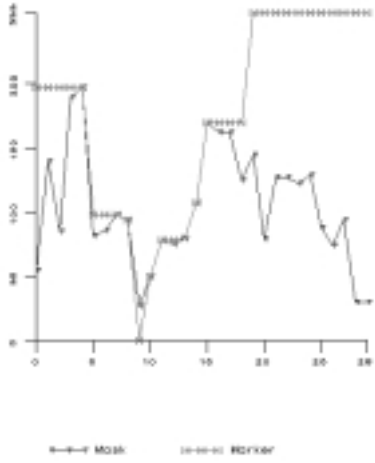

Figura 4.16: Resultado após doze iterações

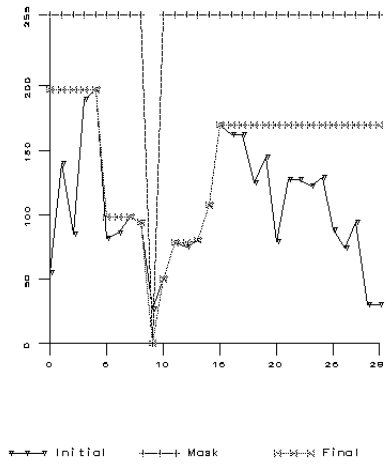

Figura 4.17: Resultado final da mudança de homotopia

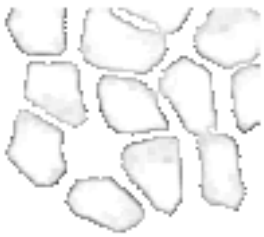

Figura 4.18: Filtragem homotópica

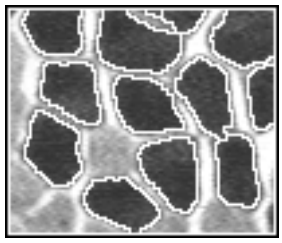

Figura 4.19: Resultado final

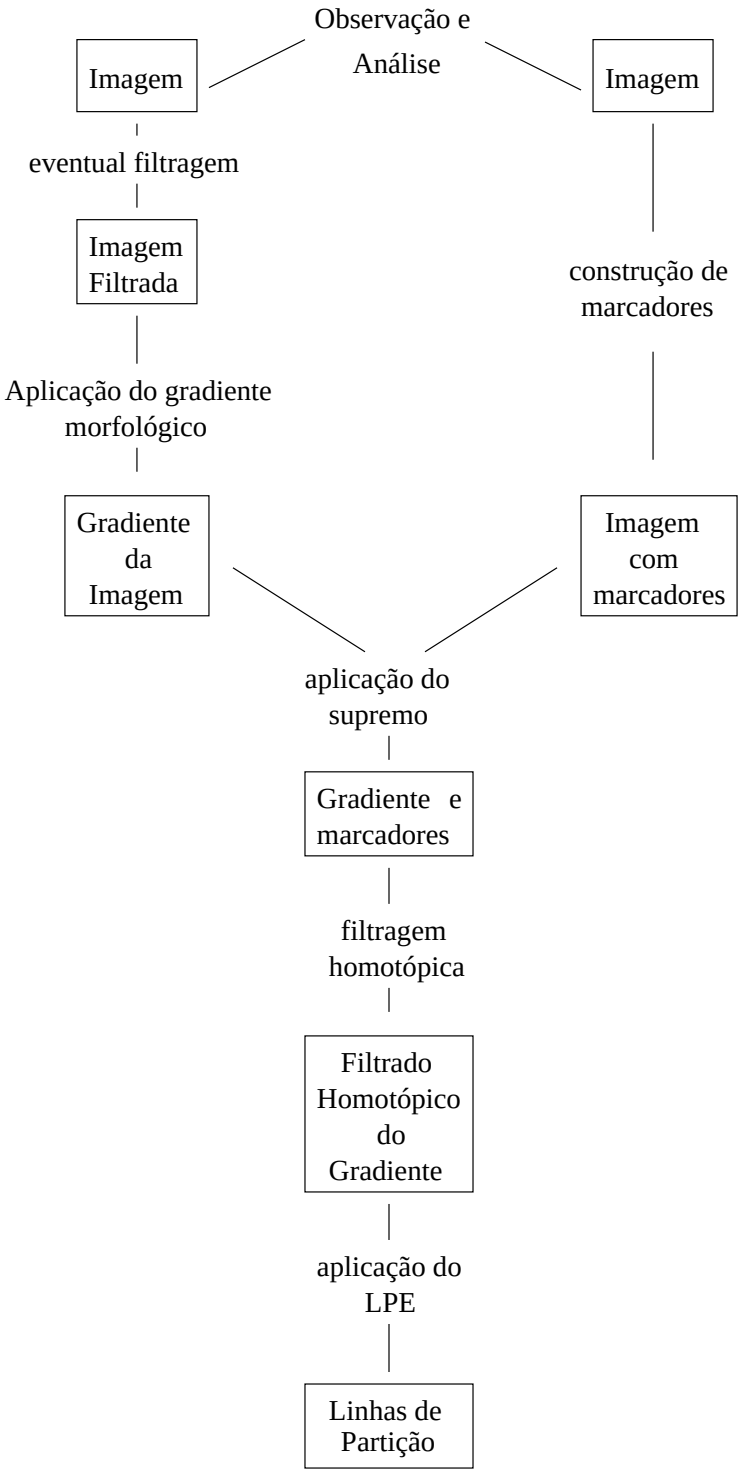

Figura 4.20: Paradigma de Beucher

4.3 Considerações finais sobre a abordagem morfológica

No capítulo 2 vimos algumas transformações úteis para segmentação de imagens e no capítulo 3 vimos que qualquer uma delas pode ser escrita em termos dos operadores elementares da Morfologia Matemática, devido ao teorema da decomposição canônica [BB93]. Assim, dado um operador clássico qualquer que segmente $\mathbf{E}$, digamos T, é possível escrever um operador usando os operadores elementares da *MM* , digamos Ψ, que é equivalente a T (fig. 4.21). Desta forma, a abordagem heurística da segmentação de imagens clássica é totalmente equivalente à abordagem morfológica. Um dos melhores exemplos desta eqüivalência é o operador threshold. Quando estudado sob o ponto de vista da *MM*, ele é uma erosão [BB93].

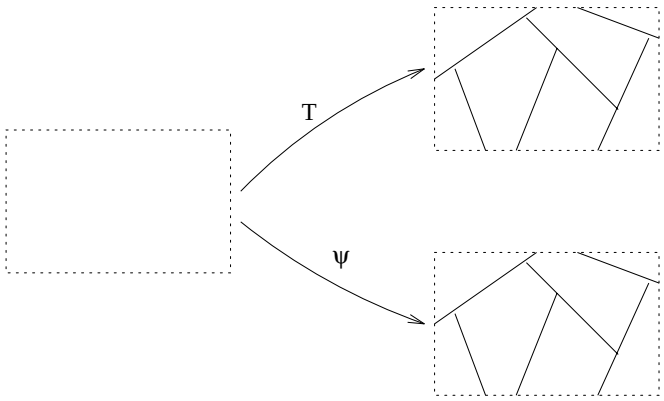

Figura 4.21: Equivalência entre as abordagens clássica e morfológica

Vimos também que existem outras abordagens mais sofisticadas que são baseadas em procedimentos estatíscos. Estas abordagens usam duas funções, uma para fazer a extração de atributos coletados sobre a imagem, digamos S, e uma para fazer a classifição desses atributos e segmentar o espaço $\mathbf{E}$, digamos ω. Quando esses atributos são coletados através de uma janela W (de maneira semelhante àquela mostrada na fig. 2.18 do capítulo 2) , a abordagem estatística clássica é equivalente à abordagem morfológica pois podemos construir um operador morfológico restrito a uma janela W, denotado Ψ^W (veja a fig. 4.22) que segmenta $\mathbf{E}$ da mesma forma que ω [Ban95].

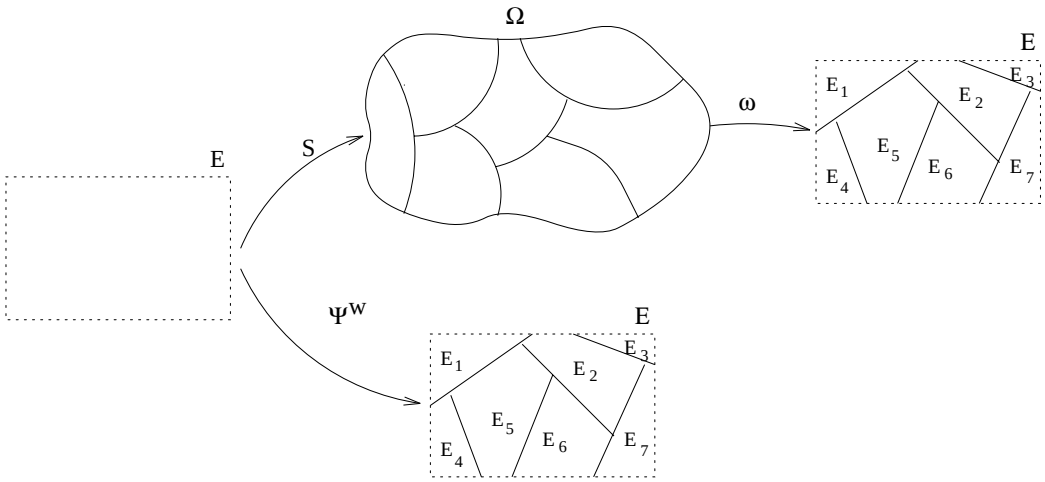

Figura 4.22: Equivalência entre as abordagens estatístico-clássicas e morfológica

Capítulo 5

Exemplos de Segmentação

Neste capítulo apresentaremos uma coletânea de exemplos reais de segmentação de imagens que foram resolvidos utilizando as técnicas e transformações da Morfologia Matemática. Muitas soluções são originais e todas elas foram desenvolvidas utilizando-se o ambiente Khoros e as ferramentas de Morfologia Matemática desenvolvidas para esse ambiente [veja apêndice A].

O objetivo deste capítulo é, em primeiro lugar, servir como um pequeno compêndio de exemplos[1] donde seja possível extrair idéias e inspirações para a solução de outros problemas de segmentação de imagens. Em segundo lugar, gostaríamos que ele servisse para mostrar que a Morfologia Matemática é rica não apenas como uma teoria em si, mas como uma poderosa ferramenta prática.

Os problemas que veremos encontram-se agrupados de acordo com a área de aplicação que os originou. Para cada grupo, apresentaremos pelo menos um exemplo cuja solução encontrada é heurística e, quando possível, um exemplo em que foi usado o Paradigma de Beucher.

Abaixo apresentamos uma pequena explicação de cada grupo de aplicações.

- **Acadêmico**

  Este grupo contém dois exemplos de segmentação cuja solução encontrada é heurística. O nome deste grupo vem do fato das imagens não terem sido tomadas de aplicações reais.

- **Indústria**

  Este grupo contém diversos exemplos de segmentação para aplicações relativas ao auxílio do controle de qualidade e automação industrial.

- **OCR**

  Este grupo contém exemplos de segmentação importantes nas aplicações de reconhecimento óptico de caracteres em imagens de textos digitalizados.

- **Biologia**

[1] Os exemplos que serão apresentados neste capítulo fazem parte da documentação "online" da MMach e encontram-se disponíveis, juntamente com a toolbox MMach, no endereço http://www.ime.usp.br/dcc/vision/

Este grupo contém dois exemplos de aplicação relativos à, respectivamente, segmentação de verminoses em amostras de sangue e de fibras de tecido muscular.

5.1 Acadêmico

Apresentaremos a seguir exemplos de projeto de operadores para a solução de problemas inventados para ilustrar técnicas importantes de segmentação morfológica.

5.1.1 Anel

A imagem da fig. 5.2 mostra uma série de objetos dispostos em forma de anel, com exceção de um disco que está centrado nele. O problema a ser resolvido é encontrar um operador que segmente os objetos que formam o anel. A formulação do problema é bastante simples, mas a solução é engenhosa.

Figura 5.1: Elemento estruturante

A idéia é apagar o disco central usando uma dilatação por um elemento estruturante em formato de um círculo centrado na origem e, cujo raio é um pouco maior do que o diâmetro do disco central (mas menor do que o raio do círculo interno do anel). Note que nenhum ponto do disco pertencerá ao resultado da dilatação pois, quando o centro do círculo estiver sobre um ponto do disco, nenhum ponto do círculo estará tocando o disco. Por outro lado, quando o centro do círculo estiver sobre um ponto de qualquer objeto do anel, nenhum de seus pontos estará tocando o disco central.

A imagem da fig. 5.1 mostra uma representação do elemento estruturante usado e a imagem da fig. 5.3 mostra o resultado da dilatação da imagem da fig. 5.2 por este círculo.

O resultado da dilatação tem interseção não vazia com todos os objetos do anel na imagem original, mas tem interseção vazia com o disco no centro do anel, que é justamente o que queríamos.

A imagem da fig. 5.4 mostra o ínfimo da imagem original com a imagem dilatada. Apesar do disco ter desaparecido, esta imagem ainda não é o resultado final, pois alguns objetos não estão completos. Por outro lado, pelo menos uma parte de cada objeto de interesse está presente na imagem, o que nos leva a fazer uma reconstrução da imagem original usando como marcadores a imagem resultante do ínfimo.

A imagem da fig. 5.5 mostra o resultado da reconstrução da imagem da fig. 5.2 pela imagem da fig. 5.4, i.e. usando como marcadores os objetos da imagem da fig. 5.2.

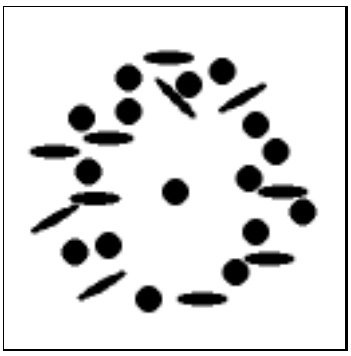

Figura 5.2: Objetos diversos

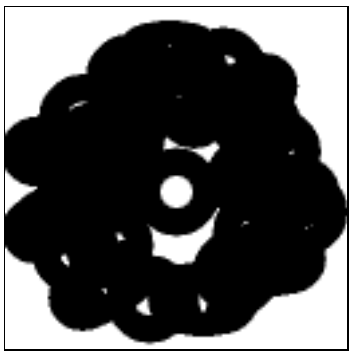

Figura 5.3: Dilatação

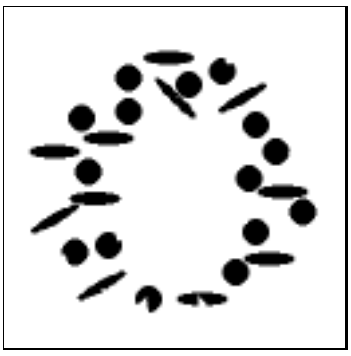

Figura 5.4: Ínfimo

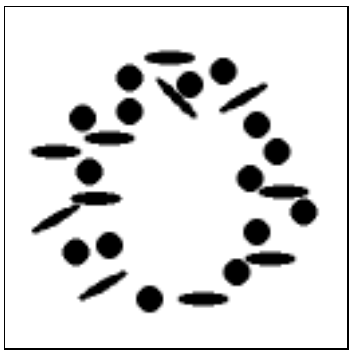

Figura 5.5: Imagem segmentada

5.1.2 Objetos Sobrepostos

A imagem da fig. 5.6 mostra uma série de objetos arredondados sobrepostos, o problema que queremos resolver é separar os objetos. As aplicações deste problema são variadas, por exemplo, a contagem de glóbulos brancos no sangue, ou contagem da quantidade ou estimativa do tamanho médio de grãos de soja, ou feijão, etc.

A imagem é binária e para resolver o problema vamos usar o operador LPE. Primeiramente vamos aplicar a função distância sobre a imagem original para conseguir um modelo dos objetos que os diferencie pela distância de seus centros às suas bordas. A imagem da fig. 5.7 mostra o resultado da função distância. Os níveis de cinza da imagem indicam as diferentes distâncias em relação às bordas dos objetos e não a relação de ordem entre elas. Isto acontece pois utilizamos uma tabela de cores aleatória para melhor visualização do resultado.

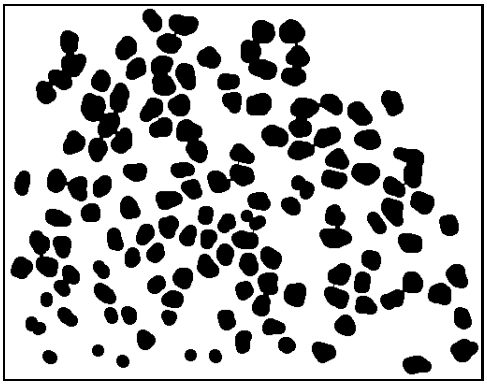

Figura 5.6: Objetos sobrepostos

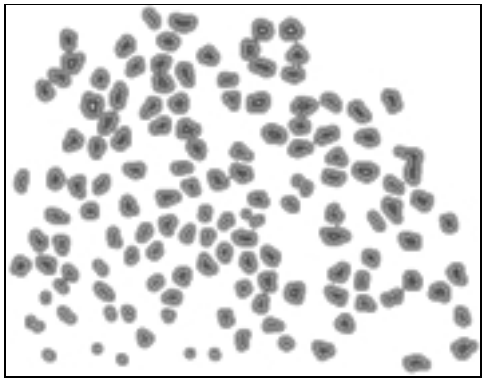

Figura 5.7: Função Distância

Em seguida temos que calcular as linhas de partição d'águas desta função distância, mas antes temos que tomar o cuidado para que haja apenas um máximo local da função distância para cada objeto na imagem. Desta forma teremos apenas um marcador por objeto. Assim, primeiramente vamos achar estes máximos locais e juntá-los quando houver mais de um por objeto. Para juntar os máximos locais basta fazer uma dilatação suficientemente grande. Como os marcadores não devem ser muito grandes para evitar um mal posicionamento das linhas de partição, faz-se em seguida uma erosão para diminuí-los. A imagem da fig. 5.8 mostra os marcadores encontrados pelo operador descrito acima. A imagem foi composta com a imagem original para melhor visualização do posicionamento dos marcadores.

Em seguida fazemos o supremo dos marcadores (fig. 5.8) com o resultado da função distância (fig. 5.7) e aplicamos o operador LPE sobre a imagem negada deste supremo. A imagem da fig. 5.9 mostra as linhas de partição d'águas encontradas. Note que as linhas não correspondem às bordas dos objetos, mas elas podem ser usadas para separá-los.

A imagem da fig. 5.10 mostra a subtração morfológica da imagem original pela imagem das linhas de partição. Desta forma podemos separar os objetos que se sobrepõe.

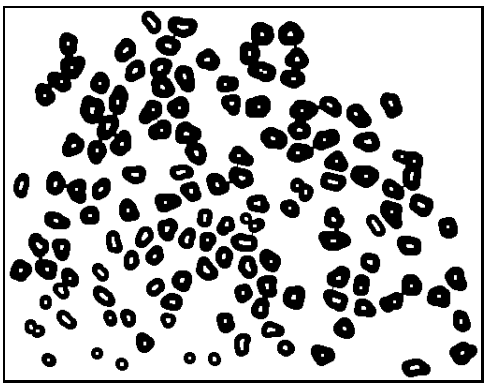

Figura 5.8: Objetos e seus marcadores

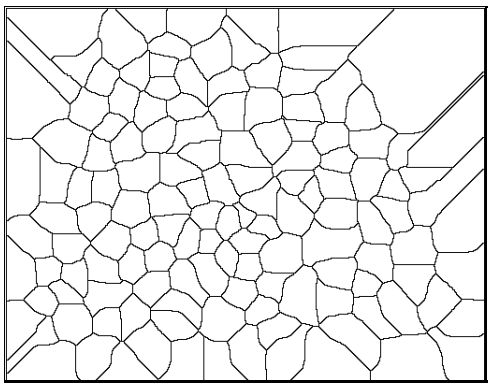

Figura 5.9: Linhas de Partição d'Águas

Como as bordas não são muito suaves, podemos melhorá-las aplicando uma abertura por um disco (usamos um disco de diâmetro sete em métrica Euclidiana) para alisá-las. A imagem da fig. 5.11 mostra o resultado final da segmentação.

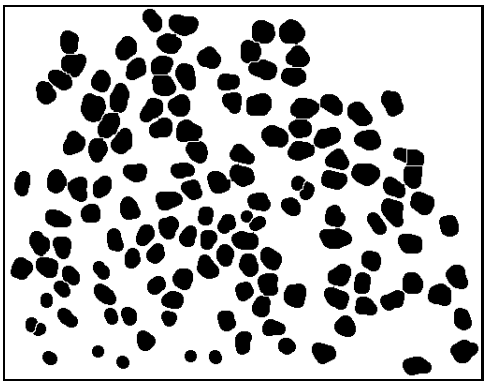

Figura 5.10: Subtração Morfológica

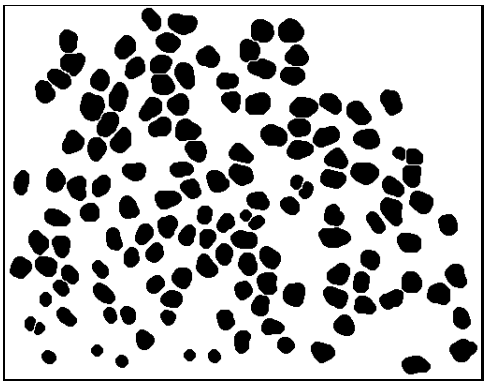

Figura 5.11: Abertura

5.2 Indústria

Nesta seção apresentaremos exemplos de processamento de imagens aplicados ao controle de qualidade na indústria.

5.2.1 PCB

A imagem da fig. 5.12 mostra uma placa de circuito impresso (PCB - "Printed Circuit Board"). O problema a ser resolvido é encontrar um operador que segmente os furos da placa onde os componentes deverão ser encaixados (montados na placa) e soldados. Uma aplicação possível para este operador é auxiliar no controle de qualidade das placas (por exemplo, as que têm menos furos do que o esperado devem ser rejeitadas).

Este é um problema relativamente simples se resolvido com os operadores morfológicos. Para resolver o problema, primeiramente aplicamos o operador fecha-buracos ("Close-holes"). A idéia é reconstruir todos os pontos do fundo da imagem que tocam a moldura. Os únicos pontos que não são reconstruídos são os pontos dos buracos das componentes conexas. O resultado da aplicação deste operador, que pode ser visto na imagem da fig. 5.13, é uma imagem em que as componentes conexas não têm buracos.

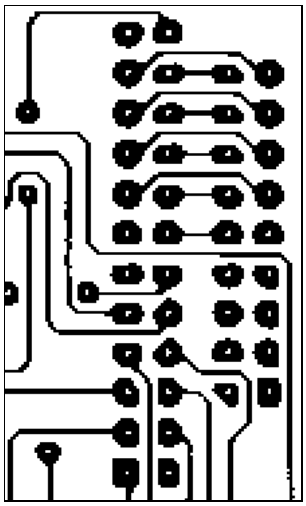

Figura 5.12: Placa de Circuito Impresso

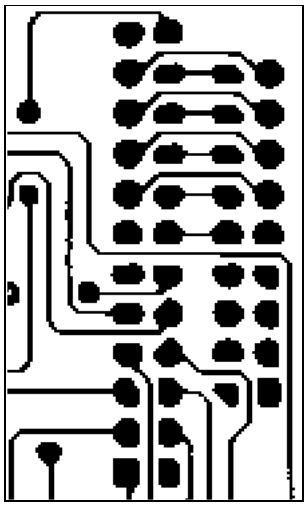

Figura 5.13: Os furos são fechados

Ao subtrairmos a imagem da fig. 5.12 da imagem da fig. 5.13, teremos a imagem formada por todos os pontos que pertencem à imagem sem buracos e não pertencem à imagem original, ou seja, a imagem correspondente aos furos.

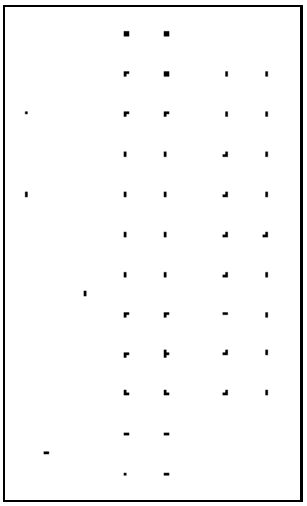

Figura 5.14: Furos da placa

5.2.2 Calculadora - botões

A imagem da fig. 5.15 mostra a parte das teclas do painel frontal de uma calculadora de bolso. Dois problemas de segmentação podem ser resolvidos sobre ela. Podemos segmentar os dígitos impressos nas teclas para um posterior reconhecimento, ou podemos segmentar as bordas das teclas. Mostraremos neste exemplo como resolver os dois problemas.

Segmentação dos caracteres

Os pontos que formam as teclas são, em média, mais escuros que os pontos fora das teclas ou os pontos dos caracteres impressos. Estes últimos, por sua vez são, em média, bem claros. Uma limiarização poderia ser tentada para segmentar os dígitos, mas o resultado não é muito bom.

A melhor maneira de segmentar os picos mais altos de uma imagem, como neste caso, é usar o operador "top hat", que é o resíduo da reconstrução da imagem original a partir de uma erosão da imagem original. A imagem da fig. 5.16 mostra o resultado após cinco erosões da imagem original por um quadrado elementar.

Figura 5.15: Imagem original

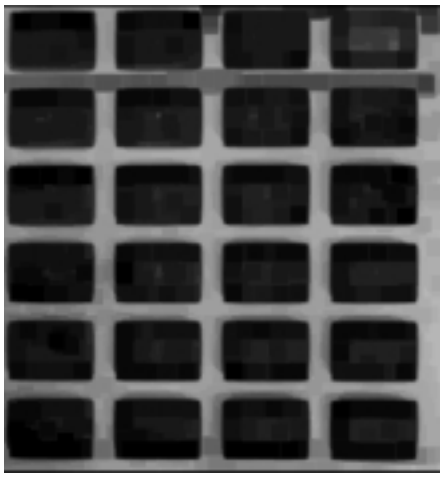

Figura 5.16: Imagem erodida - 5

A imagem da fig. 5.17 mostra o resultado da reconstrução da imagem original pela imagem da fig. 5.16. O resíduo ou subtração da imagem original pela imagem da fig. 5.17 é a imagem em níveis de cinza dos dígitos (veja fig. 5.18). Uma simples limiarização resolve a segmentação dos dígitos, como pode ser visto na imagem da fig. 5.19.

Segmentação das bordas das teclas

Vamos abordar a segmentação das bordas das teclas pelo Paradigma de Beucher, que sabemos ser uma técnica adequada para resolver este problema.

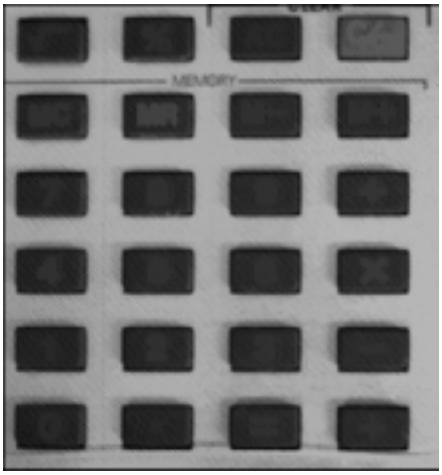

Figura 5.17: Reconstrução

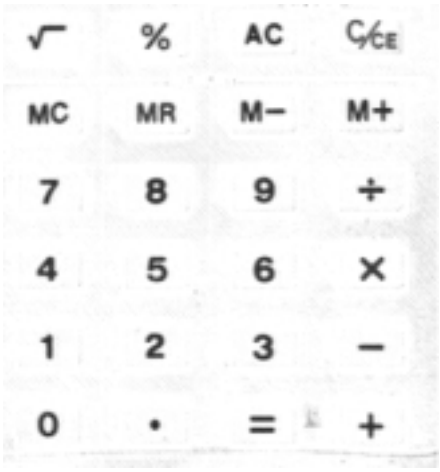

Figura 5.18: Imagem do resíduo

Como cada tecla tem um caractere impresso, vamos usar estes dígitos como marcadores internos para as teclas. Para isso, primeiramente vamos dilatar a imagem dos dígitos 3 vezes por um quadrado elementar para tornar os marcadores conexos. O resultado da dilatação pode ser visto na imagem da fig. 5.20.

Figura 5.19: Dígitos segmentados

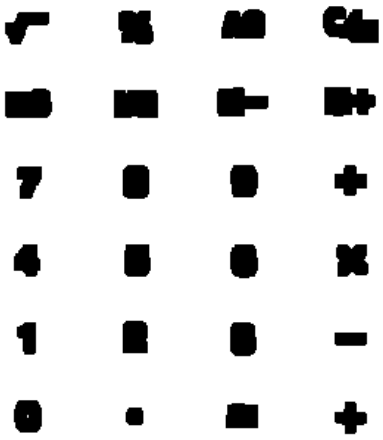

Figura 5.20: Dilatação dos dígitos

Para construir os marcadores externos, vamos aumentar o tamanho da imagem, colocar uma moldura e calcular a zona de influência geodésica (SKIZ) dos objetos da imagem. A imagem da fig. 5.21 mostra o SKIZ da imagem da fig. 5.20. A imagem da fig. 5.22 mostra o supremo das imagens dos marcadores internos e externos para as teclas.

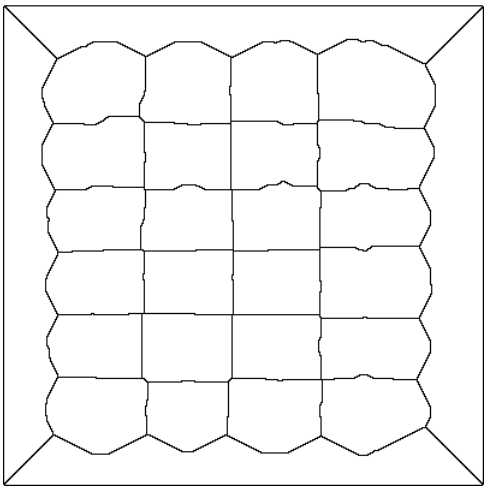

Figura 5.21: SKIZ da dilatação

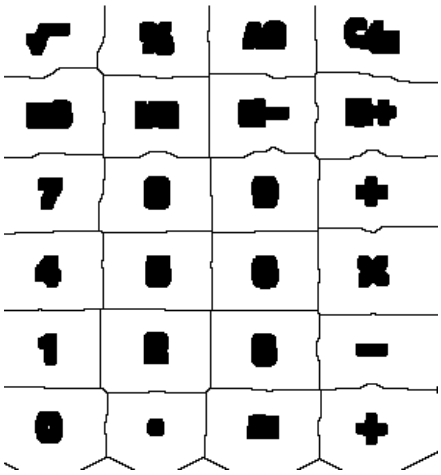

Figura 5.22: Marcadores

Para aplicar o Paradigma de Beucher vamos, primeiramente, calcular o gradiente da imagem original. A imagem da fig. 5.23 mostra o resultado do gradiente. A imagem foi invertida para que os detalhes pudessem ser vistos no papel.

Em seguida, fazemos a mudança de homotopia do gradiente da imagem original a partir da imagem dos marcadores (fig. 5.22). A imagem da fig. 5.24 mostra o resultado do operador.

Figura 5.23: Gradiente- invertido

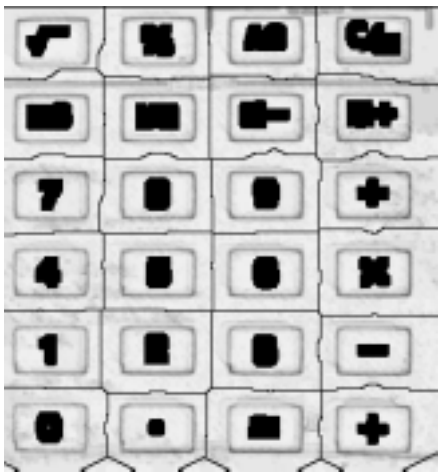

Figura 5.24: Mudança da homotopia

Por último, aplica-se o operador LPE sobre a imagem da fig. 5.24 para achar as linhas de partição d'águas da imagem modificada. A imagem da fig. 5.25 mostra o resultado do operador LPE. As linhas escuras correspondem às linhas de partição de águas e cada tom de cinza indica

uma bacia de captação, que corresponde a uma tecla.

A imagem da fig. 5.26, obtida através de uma simples limiarização da fig. 5.25, mostra apenas as linhas de partição.

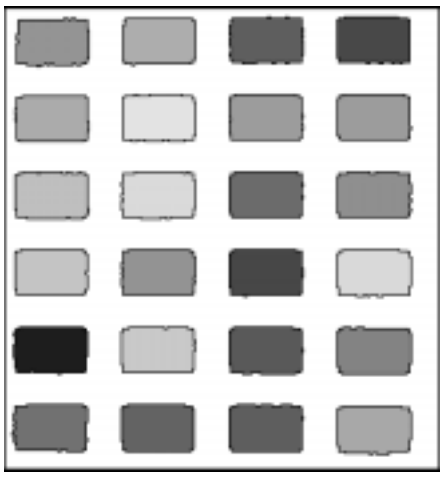

Figura 5.25: Watershed mais regiões

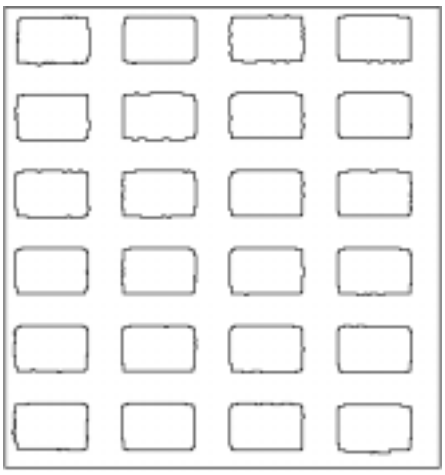

Figura 5.26: LPE

5.2.3 Segmentação de Arestas para Aplicações Robóticas

O problema que apresentaremos a seguir ilustra bem a generalidade do Paradigma de Beucher. A imagem da fig. 5.27 mostra um sólido visto por uma câmera.

Usualmente, um robô que manipula objetos sólidos 3D tem como entrada imagens 2D destes objetos. Para posicionar as garras do robô para que este pegue um objeto é importante saber a posição das arestas do objeto. Isso pode ser feito por um pré-processador que extrai as arestas dos objetos de interesse das imagens de entrada.

A imagem da fig. 5.28 mostra o resultado do operador gradiente aplicado à imagem da fig. 5.27. A imagem foi invertida para melhor visualização dos detalhes no papel.

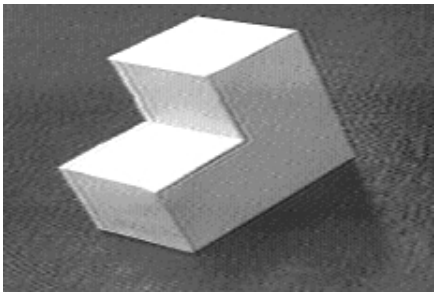

Figura 5.27: Imagem original

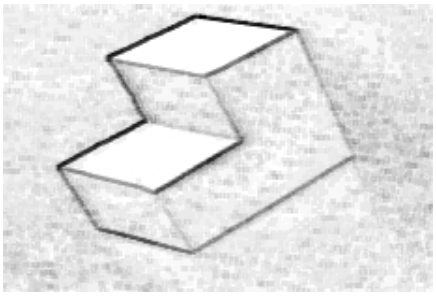

Figura 5.28: Gradiente invertido

A imagem da fig. 5.29 mostra o resultado do operador LPE aplicado sobre a imagem do gradiente. Este é mais um exemplo de que o operador LPE não deve ser aplicado sobre a imagem do gradiente sem a filtragem homotópica, que faremos a seguir.

Para encontrar os marcadores para as faces do sólido usamos primeiramente três limiarizações diferentes (uma para as duas faces mais claras, uma para a face em forma de L e para a face mais escura inferior e uma para a face mais escura superior). Em seguida, para cada limiarização escolhemos uma erosão que tornasse o marcador único para cada face. Para o marcador externo, fizemos uma limiarização que segmentasse todo o bloco e depois aplicamos um filtro composto por uma abertura por um disco de diâmetro 8, uma dilatação por um disco de diâmetro 10 e uma erosão por um quadrado elementar. A imagem da fig. 5.30 mostra o supremo dos marcadores internos e externos obtidos pelas limiarizações e filtragens morfológicas da imagem original. Note que todas as faces contém um marcador, para que todas as arestas sejam encontradas. Isto foi estabelecido pelo problema proposto.

A imagem da fig. 5.31 mostra o resultado da mudança de homotopia da imagem do gradiente morfológico usando como marcadores a imagem da fig. 5.30. A imagem foi invertida para melhor visualização dos detalhes no papel. Note que a imagem está mais suave do que a imagem do gradiente.

A imagem da fig. 5.32 mostra o resultado do operador LPE aplicado na imagem da fig. 5.31. A imagem da fig. 5.33 mostra o resultado de uma limiarização para segmentar apenas as linhas

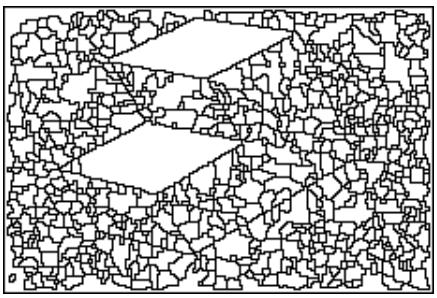

Figura 5.29: Watershed frustado

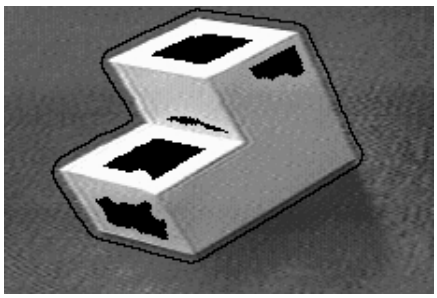

Figura 5.30: Marcadores

de partição de águas da imagem.

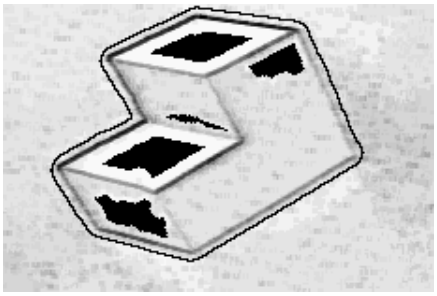

Figura 5.31: Mudança de homotopia

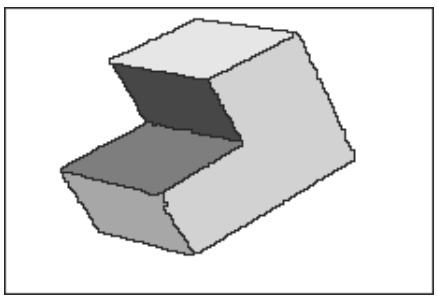

Figura 5.32: Watershed

A imagem da fig. 5.34 mostra o resultado da composição da imagem das linhas de partição com a imagem original.

A composição destes operadores resulta em um operador para segmentar arestas de imagens 2D bastante robusto, como pode ser visto nos exemplos que seguem. A imagem da fig. 5.35 e da fig. 5.37, quando usadas como entrada deste operador, resultam na imagem da fig. 5.36 e fig. 5.38, respectivamente.

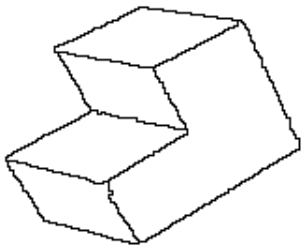

Figura 5.33: LPE

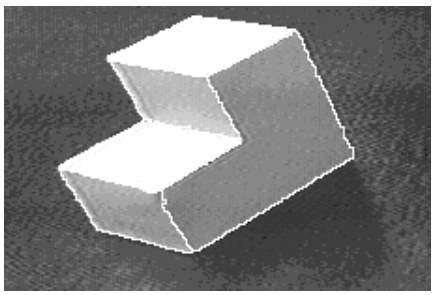

Figura 5.34: Composição

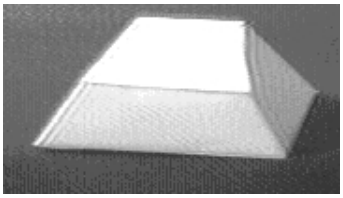

Figura 5.35: Outro bloco

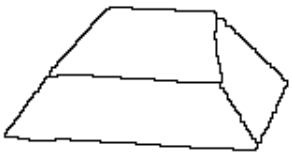

Figura 5.36: LPE

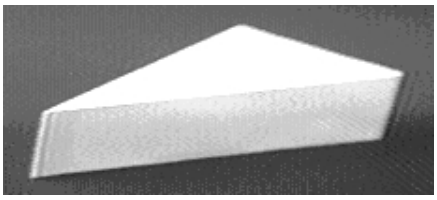

Figura 5.37: Outro bloco

Figura 5.38: LPE

5.2.4 Metal

Uma aplicação muito importante na indústria é a detecção de fissuras em laminados e peças de metal, pois ela é de fundamental importância para fins de segurança e qualidade do produto.

Existem diversas técnicas para detectar fissuras em lâminas de metal, uma delas é a inspeção de defeitos na estrutura cristalina em imagens tomadas com uso de raio-X. A imagem da fig. 5.39 mostra uma imagem binária deste tipo.

A solução para este problema é complexa e totalmente heurística. O objetivo principal é encontrar as regiões da imagem onde há falhas na conexidade dos objetos alongados.

Primeiramente, vamos achar o esqueleto homotópico da imagem para ter um modelo dos objetos que preserva o comprimento. A imagem da fig. 5.40 mostra o resultado deste operador.

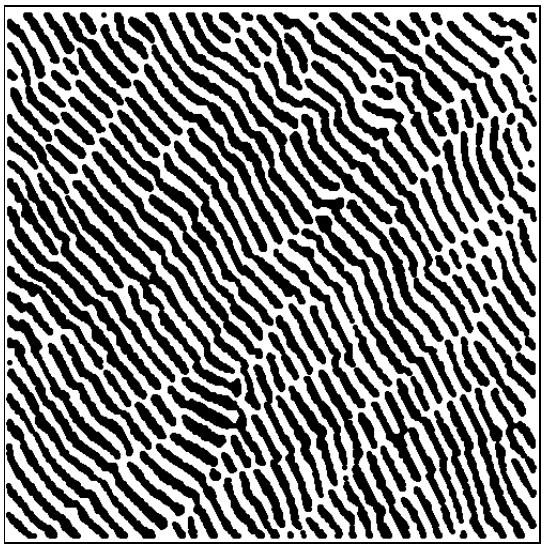

Figura 5.39: Lâmina metálica

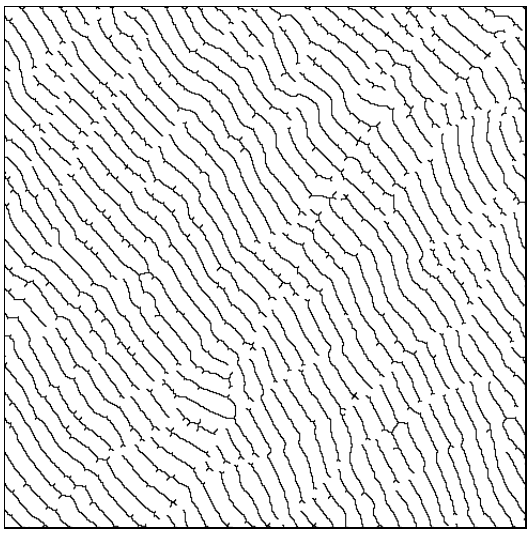

Figura 5.40: Esqueleto homotópico

Os pontos extremos do esqueleto homotópico contém pontos que estão próximos das regiões de falha. Para encontrá-los, vamos aplicar o operador sup-gerador. A imagem da fig. 5.41 mostra o resultado do operador sup-gerador aplicado sobre imagem da fig. 5.40 para achar os pontos extremos.

O resultado contém os pontos de interesse, mas muitos outros também, devido às "barbas"[2] do esqueleto. O próximo objetivo será escolher os pontos importantes da imagem da fig. 5.41.

A imagem da fig. 5.42 mostra o resultado do operador de afinamento para "barbear" ("podar") o esqueleto, composto com uma borda grossa para tornar as linhas que tocam as bordas uma só componente conexa. Os pontos extremos desta imagem são pontos importantes pois com eles podemos detectar as falhas de conexidade dos objetos. Por outro lado, eles são poucos para

[2]barba é um nome usual para designar as pequenas pontas que aparecem entre os extremos do esqueleto

construir um marcador para essas regiões, por isso que os usaremos para escolher pontos na imagem da fig. 5.41.

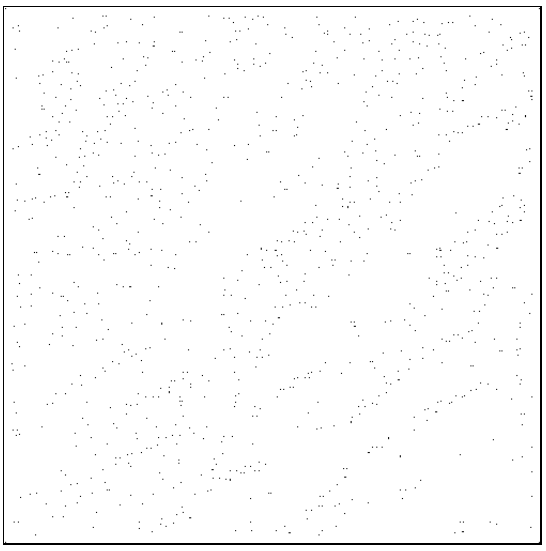

Figura 5.41: Pontos extremos

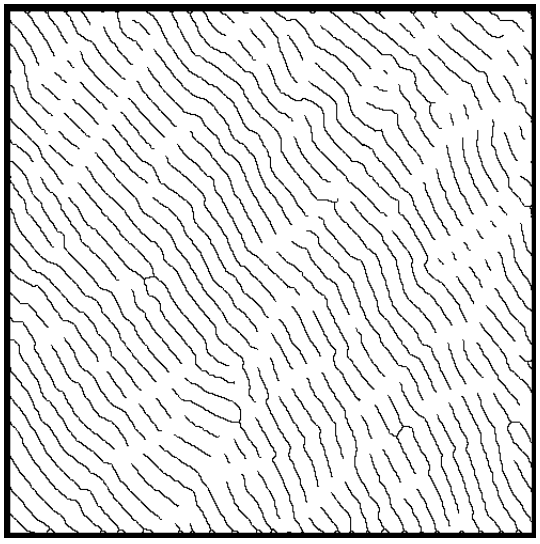

Figura 5.42: Esqueleto barbeado

A imagem da fig. 5.43 mostra o resultado de um operador morfológico que escolhe da imagem da fig. 5.41 os pontos próximos às regiões de falha. Este operador é construído pela composição de uma dilatação dos pontos extremos da fig. 5.42 seguido de um ínfimo com a imagem da fig. 5.41.

A imagem da fig. 5.44 mostra o resultado do supremo entre a imagem da fig. 5.42, sem a borda, com a imagem da fig. 5.43. O próximo objetivo será construir um objeto conexo, a partir destes pontos, que preencha aproximadamente as regiões de interesse.

A imagem da fig. 5.45 mostra o resultado do operador SKIZ que calcula a zona de influência geodésica sobre o resultado de uma dilatação da imagem da fig. 5.44.

A imagem da fig. 5.46 mostra a reconstrução do inverso da imagem da fig. 5.45 usando como marcadores a imagem da fig. 5.42. Com isso conseguimos "pintar" as possíveis regiões de falha.

Nosso objetivo agora será segmentar as regiões onde podem ocorrer falhas, que é um subconjunto das regiões da imagem da fig. 5.46. Para isso, vamos primeiramente segmentar as linhas do SKIZ que não pertencem à essas regiões e também juntar um pouco mais as regiões maiores da imagem. Isso é feito por um fechamento (para aglomerar as regiões próximas) seguido de uma abertura por um elemento estruturante maior que a espessura da linha que queremos segmentar. A imagem da fig. 5.47 mostra o resultado desta composição.

Em seguida, vamos aplicar o esqueleto homotópico novamente para ter um modelo das regiões segmentadas. A imagem da fig. 5.48 mostra o resultado deste esqueleto.

Nem todas as regiões onde ocorrem as falhas de conexidade são importantes para a descoberta das regiões mecanicamente fracas. As regiões importantes são aquelas onde há mais de dois objetos

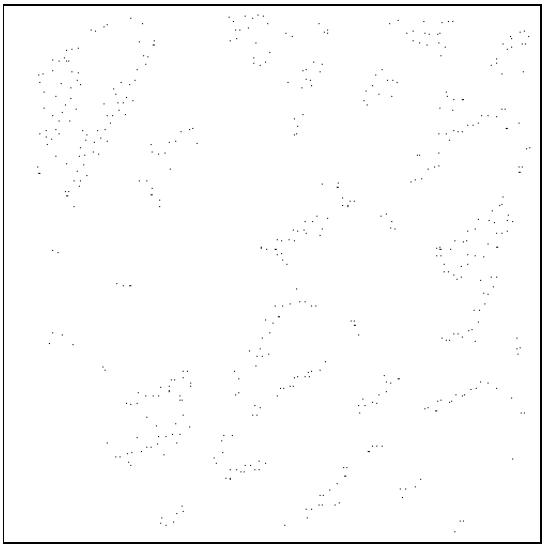

Figura 5.43: Pontos escolhidos

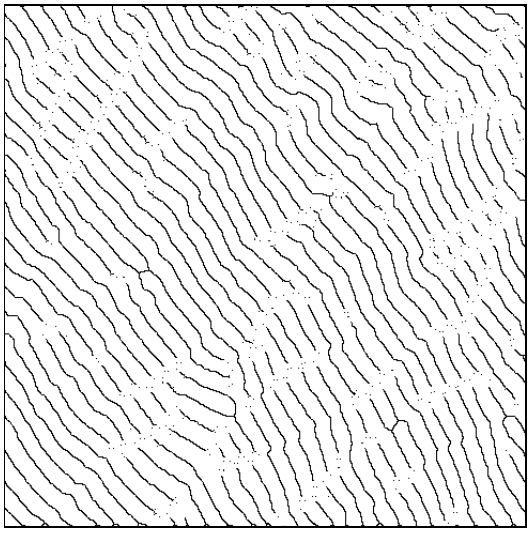

Figura 5.44: Composição

desconexos em seguida. Para segmentar estas regiões pouco importantes faremos um filtro que apague essas partes.

Inicialmente faz-se um emagrecimento dos objetos pelos extremos para eliminar os objetos menores. Em seguida acham-se os pontos extremos desses objetos e os dilatamos por um disco suficientemente grande para o resultado conter os pontos extremos reais dos objetos de interesse do esqueleto homotópico. Esses pontos extremos que estão contidos nos discos grandes são dilatados por um disco pequeno para "protegê-los" do operador de emagrecimento pelos extremos. A imagem da fig. 5.49 mostra um passo intermediário onde aparecem os pontos extremos do esqueleto homotópico já "protegidos".

Finalmente, aplica-se um emagrecimento pelos extremos para apagar os objetos menores do esqueleto homotópico sem apagar os maiores.

A imagem da fig. 5.50 mostra o resultado do filtro para segmentar as partes dos esqueletos das regiões que podem ser fracas mecanicamente.

A imagem da fig. 5.51 mostra a composição do resultado com a imagem original.

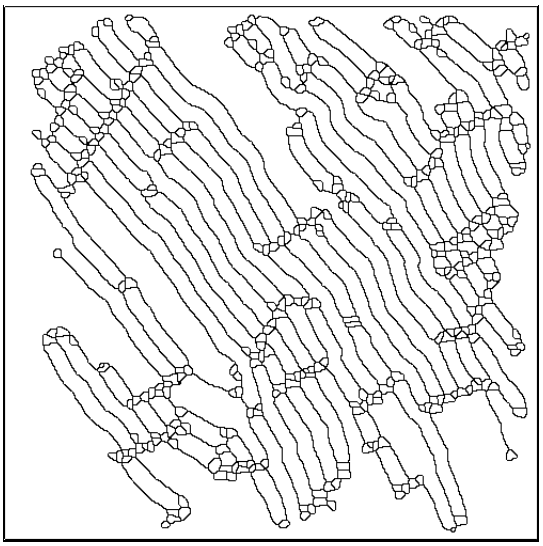

Figura 5.45: SKIZ

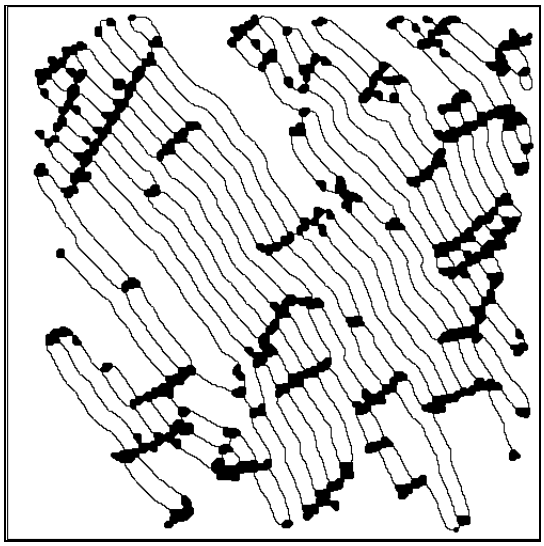

Figura 5.46: Reconstrução

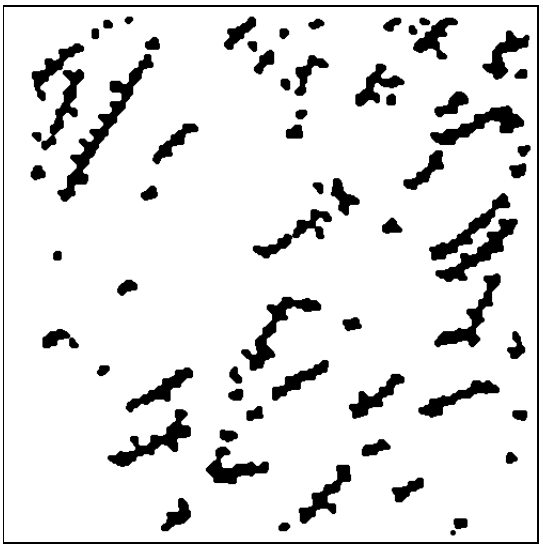

Figura 5.47: Regiões

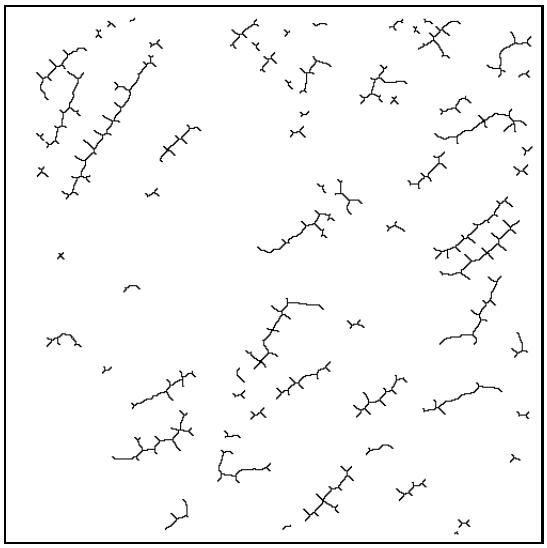

Figura 5.48: Esqueleto homotópico

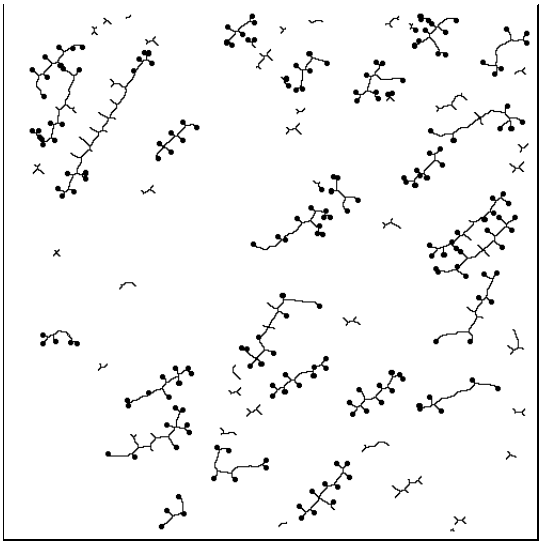

Figura 5.49: Passo intermediário

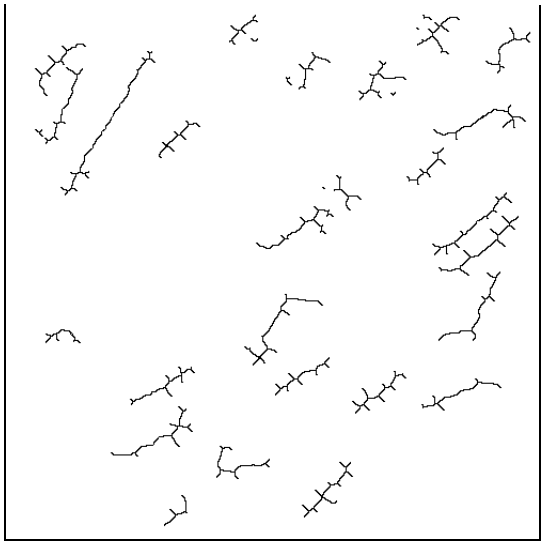

Figura 5.50: Resultado final

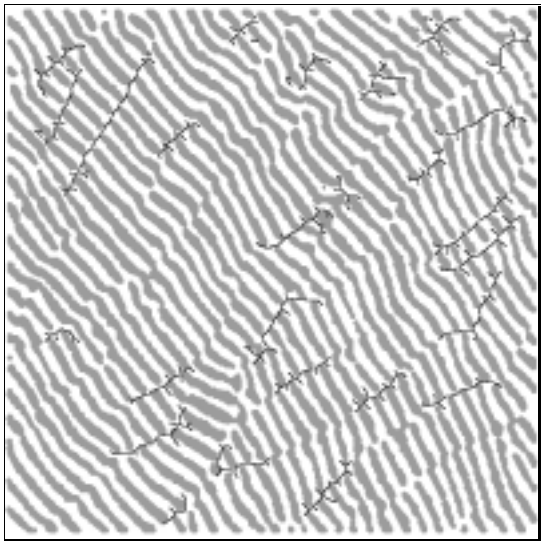

Figura 5.51: Composição com a imagem original

5.3 OCR

Nesta seção mostraremos alguns exemplos de como os operadores morfológicos podem ser usados para a segmentação de estruturas em imagens de textos digitalizados. Todos os problemas que apresentaremos aqui foram resolvidos heuristicamente.

Os bons programas de OCR[3] usam segmentações do tipo que apresentaremos como parte do processo de reconhecimento das estruturas contidas num texto, por exemplo, parágrafos e seus tipos, palavras, letras, ilustrações e etc. Estas informações podem ser usadas tanto no reconhecimento dos caracteres (i.e., na transformação da informação da forma de uma letra na informação do seu código interno no computador), como na formatação do texto no arquivo de saída.

5.3.1 Segmentação de Palavras

Mostraremos nesta seção a aplicação de um operador morfológico simples para a segmentação das palavras (conjunto de caracteres alinhados horizontalmente, separados por um espaço em branco para indicar a próxima palavra) na imagem de um texto impresso.

A imagem da fig. 5.52 mostra um texto digitalizado e limiarizado. O problema que queremos resolver é achar e distinguir cada palavra do texto. Este é um problema importante para, durante o reconhecimento dos caracteres, determinar como os caracteres reconhecidos estão agrupados em palavras.

I - INTRODUÇÃO

A **Terapia de Linguagem** com crianças *psicóticas* é constantemente permeada por dificuldades relativas a fatores emocionais intrínsecos à patologia. Uma vez que, nela estão envolvidas as manifestações patológicas da desestruturação do ego do paciente.

Muitas vezes, existem resistências ao contato e a interação, **mecanismos psicóticos** de defesa, frente aos quais o terapeuta sente-se um tanto impotente.

Figura 5.52: Texto digitalizado

Pela especificação do problema sabemos que palavras são conjuntos de objetos (caracteres) separados por brancos, i.e., a diferença da distância entre as palavras e a distância entre os caracteres na imagem é que determina as palavras do texto. A idéia que usaremos será juntar os caracteres próximos via dilatações.

A imagem da fig. 5.53 mostra o resultado da aplicação de dilatações horizontais (i.e., dilatações por um elemento estruturante do tipo linha horizontal) aplicadas à imagem da fig. 5.52. A quantidade de dilatações necessárias é diretamente proporcional à distância entre os caracteres da imagem.

Após tornarmos os objetos que formam cada palavra da imagem um objeto conexo, vamos

[3] Os programas de OCR ("Optical Character Recognition") transformam uma imagem digitalizada de um texto num arquivo que pode ser lido e editado por um editor de textos.

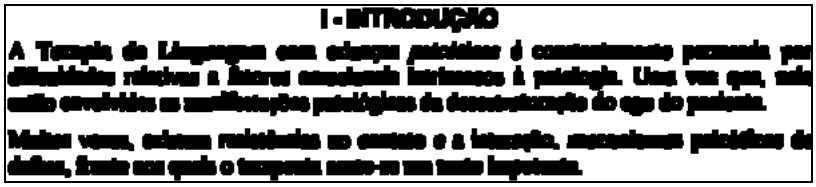

Figura 5.53: Modelo das palavras

fazer a rotulação da imagem para que cada objeto receba um rótulo diferente. A imagem da fig. 5.54 mostra o resultado da rotulação. Os níveis de cinza indicam os rótulos recebidos pelas componentes.

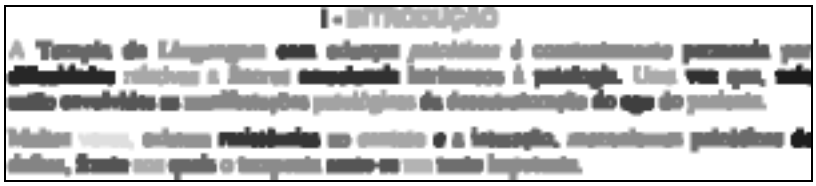

Figura 5.54: Rotulação das palavras

A imagem da fig. 5.55 mostra o ínfimo entre a imagem rotulada, mencionada anteriormente, e a imagem original. Os níveis de cinza de cada palavra indicam os rótulos que as palavras receberam.

Figura 5.55: Palavras do texto rotuladas

5.3.2 Segmentação de Parágrafos

Mostraremos nesta seção uma aplicação semelhante à anterior de um operador morfológico para a segmentação dos parágrafos de um texto. Os parágrafos que iremos segmentar são conjuntos de linhas (que por sua vez são conjuntos de palavras alinhadas) que são separados por uma linha em branco para indicar o próximo parágrafo.

A imagem da fig. 5.52 será usada novamente como entrada para a segmentação dos parágrafos.

Pela especificação, um parágrafo é um conjunto de linhas e para separar um parágrafo usa-se uma linha em branco. Usando a mesma idéia de juntar objetos que estão próximos, vamos juntar todos os caracteres que estejam na mesma linha.

A imagem da fig. 5.56 mostra o resultado de centenas de dilatações horizontais para construir um modelo das linhas do texto.

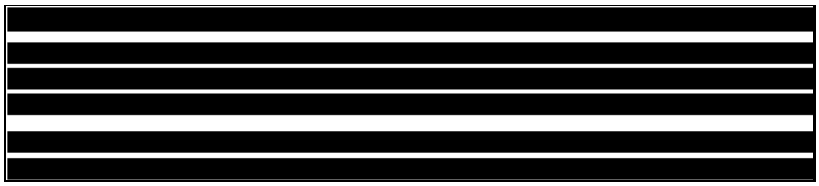

Figura 5.56: Modelo das linhas do texto

Em seguida, vamos juntar todas as linhas que estejam próximas via dilatações verticais (i.e., dilatações por um elemento estruturante do tipo linha vertical). A imagem da fig. 5.57 mostra o resultado após um número suficiente de dilatações para juntar as linhas que estão no mesmo parágrafo.

Figura 5.57: Modelo dos parágrafos do texto

Para distinguir os parágrafos vamos rotulá-los com o operador de rotulação ("labeling"). A imagem da fig. 5.58 mostra o resultado da rotulação onde os três níveis de cinza da imagem indicam a existência de três parágrafos no texto.

Figura 5.58: Modelo rotulado dos parágrafos

A imagem da fig. 5.59 mostra o resultado do ínfimo da imagem da fig. 5.52 com a imagem da fig. 5.58, que é o resultado do problema.

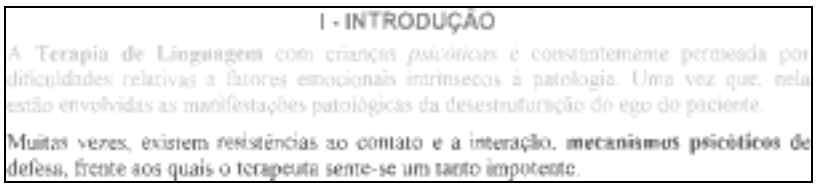

Figura 5.59: Parágrafos rotulados

5.3.3 Segmentação de Tipos de Parágrafos

Mostraremos nesta seção como podemos segmentar os parágrafos pelos seus tipos. Os tipos que iremos segmentar são parágrafos centralizados e parágrafos justificados. Idéias semelhantes às que apresentaremos podem ser usadas para a segmentação dos parágrafos alinhados à esquerda ou à direita.

A imagem da fig. 5.52 será usada novamente como entrada para a segmentação dos tipos de parágrafos.

A idéia desta segmentação é distinguir as linhas do texto que estão centralizadas das que estão justificadas. Para isso é preciso construir marcadores para a lateral direita e para o centro do texto. As linhas que não tocam o marcador da lateral direita mas tocam o marcador central são linhas que fazem parte do conjunto dos parágrafos centralizados.

A imagem da fig. 5.56 mostra o resultado de dezenas de dilatações horizontais para construir um modelo das linhas do texto.

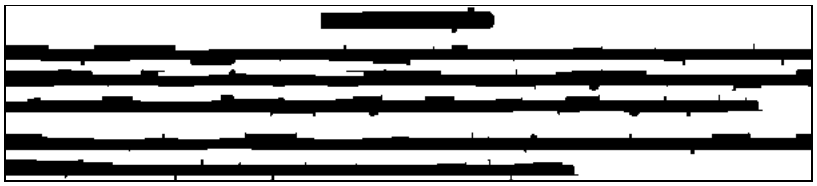

Figura 5.60: Modelo simplificado das linhas

A imagem da fig. 5.61 mostra um marcador construído por dilatações horizontais da borda direita da imagem.

A imagem da fig. 5.62 mostra o resultado da reconstrução da imagem da fig. 5.56 a partir do ínfimo entre a imagem da fig. 5.60 com a imagem da fig. 5.61, i.e., são reconstruídas as linhas que tocam a margem direita.

As linhas que pertencem aos parágrafos centralizados não tocam o marcador da direita (pela especificação acima) assim, elas devem pertencer ao resíduo da imagem original em relação à esta última.

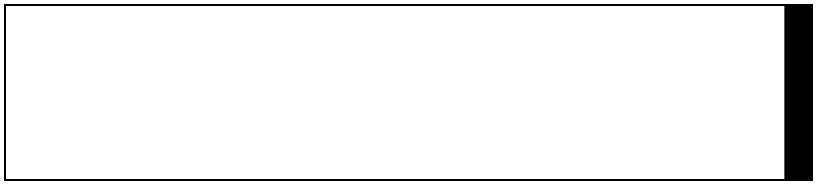

Figura 5.61: Marcador para a direita do texto

Figura 5.62: Linhas que tocam a margem direita

A imagem da fig. 5.63 mostra o resíduo da imagem da fig. 5.60 em relação à imagem da fig. 5.62.

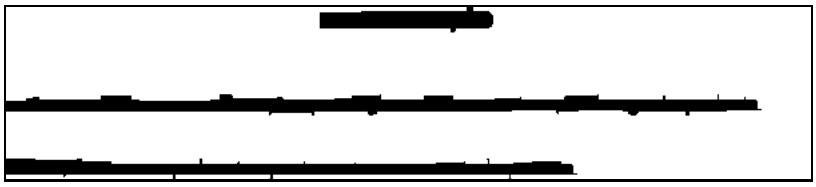

Figura 5.63: Resíduo

Para determinar quais linhas tocam o marcador central podemos primeiramente calcular seus centróides, que podem ser obtidos por um esqueleto por afinamento [BB94]. A imagem da fig. 5.64 mostra o resultado do centróide das linhas da imagem do resíduo. Os pontos foram dilatados para melhor visualização no papel.

As linhas centralizadas não precisam estar exatamente no centro horizontal da imagem, uma pequena variação em relação ao centro é aceitável, já que uma linha pode diferenciar de tamanho em relação à outra. A imagem da fig. 5.65 mostra um marcador central para o texto.

Em seguida, podemos fazer o ínfimo da imagem da fig. 5.65 com a da fig. 5.64 e usar esse resultado como marcador para a reconstrução da imagem da fig. 5.63. A imagem da fig. 5.67 mostra o resultado dessa reconstrução.

O ínfimo entre a imagem original e as linhas centralizadas recupera os parágrafos centralizados. Os outros parágrafos são justificados, já que o texto tem apenas estes dois tipos de parágrafos.

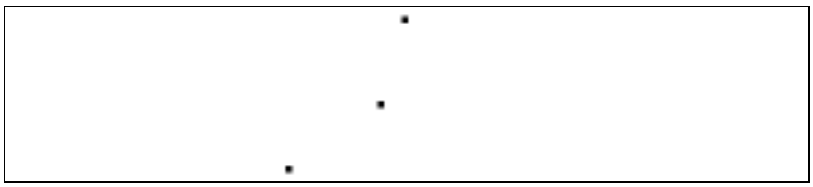

Figura 5.64: Centróide das linhas

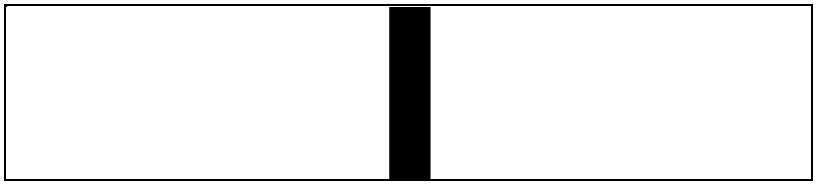

Figura 5.65: Marcador central

A imagem da fig. 5.68 mostra o resultado final da segmentação de parágrafos pelo seu tipo. Os níveis de cinza indicam o tipo de cada parágrafo.

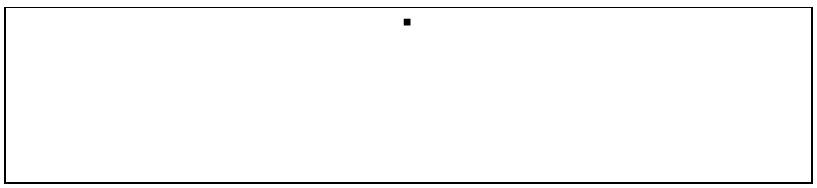

Figura 5.66: Ínfimo

Figura 5.67: Linhas reconstruídas

Figura 5.68: Parágrafos rotulados de acordo com seu tipo

5.4 Biologia

5.4.1 Danaus

A imagem fig. 5.69 mostra uma imagem microscópica de uma amostra de sangue onde podem-se ver duas larvas do transmissor da *filarioses*[4] (manchas escuras e finas, semelhantes a minhocas). O problema a ser resolvido é o de encontrar um operador que as segmente da imagem. Note que o problema não é simples dado que existem outras manchas escuras semelhantes às larvas, o que dificulta a segmentação da imagem.

A solução encontrada para problema é totalmente heurística. Vamos explicá-la dividindo o problema em diversos problemas menores. Nosso primeiro objetivo será separar as partes mais escuras da imagem. Para isso, primeiramente vamos aplicar um fechamento por reconstrução [BB94], que é a composição de uma dilatação com uma reconstrução dual. A imagem da fig. 5.70 mostra o resultado do fechamento por reconstrução da imagem da fig. 5.69 por um quadrado elementar.

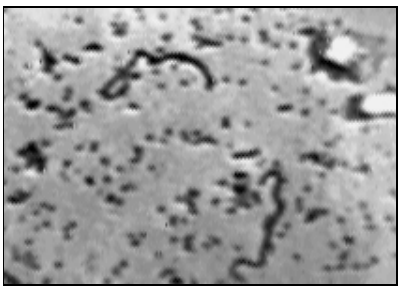

Figura 5.69: Transmissores da Filariose

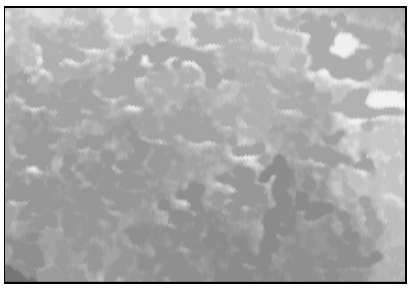

Figura 5.70: Fechamento por Reconstrução

Em seguida faremos uma subtração da imagem da fig. 5.69 pela imagem da fig. 5.70 para conseguir apenas as manchas mais escuras da imagem. A imagem da fig. 5.71 mostra o resultado desta subtração. A imagem foi invertida para que os detalhes pudessem ser visualizados no papel.

Esta composição é preferível ao threshold pois segmenta todas as partes escuras da imagem independentemente do nível de cinza da região em que elas estejam imersas.

Aplicando uma limiarização na imagem da fig. 5.71 resulta uma imagem binária correspondente às manchas. A imagem da fig. 5.72 mostra o resultado dessa limiarização. Note que a principal característica que diferencia as larvas dos outros objetos é o comprimento. Isso sugere uma abordagem semelhante à de um exemplo do capítulo 4 onde mostramos como segmentar objetos de comprimentos diferentes.

Nosso próximo subobjetivo será segmentar as larvas (que são mais longas) dos outros objetos. Para isso, primeiramente vamos aplicar o operador esqueleto por emagrecimento ("skeleton by

[4]Larvas que transmitem a elefantíase

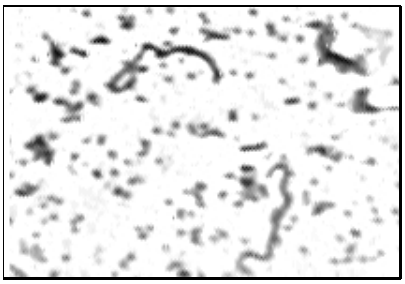

Figura 5.71: Inversão da subtração

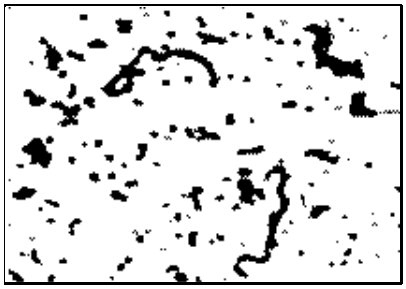

Figura 5.72: Threshold

thinning") com elementos estruturantes tais que o resultado seja o esqueleto homotópico da imagem. O resultado será um modelo dos objetos que preserva o comprimento.

A imagem da fig. 5.73 mostra o esqueleto dos objetos da imagem da fig. 5.72.

Para segmentar os objetos mais longos vamos aplicar um afinamento controlado ("N-Thinning") na imagem da fig. 5.73. A imagem da fig. 5.74 mostra o resultado do afinamento.

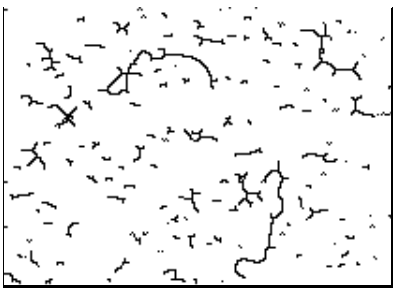

Figura 5.73: Esqueleto

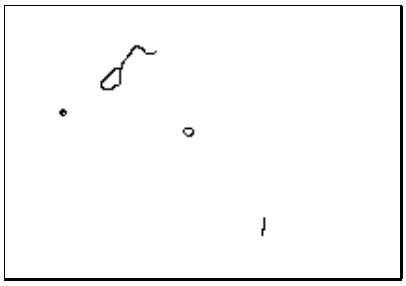

Figura 5.74: Esqueleto "emagrecido"

Note que as linhas que sobraram na imagem fazem parte das larvas, assim, se usarmos estas linhas como marcadores para as larvas, podemos segmentá-las da imagem.

O próximo subobjetivo é segmentar as linhas dos objetos arredondados. Isso pode ser feito aplicando-se mais uma vez o esqueleto por emagrecimento para podar a imagem da fig. 5.74. A imagem da fig. 5.75 mostra o resultado da poda do esqueleto.

Os marcadores para as larvas serão obtidos subtraindo-se da imagem da fig. 5.74 o resultado desta poda. A imagem da fig. 5.76 mostra este resíduo.

Para terminar a segmentação das larvas, basta fazer a reconstrução da imagem da fig. 5.72 usando como marcador a imagem da fig. 5.76.

A imagem da fig. 5.77 mostra o resultado da reconstrução e a imagem da fig. 5.78 mostra a composição da imagem segmentada com a imagem original.

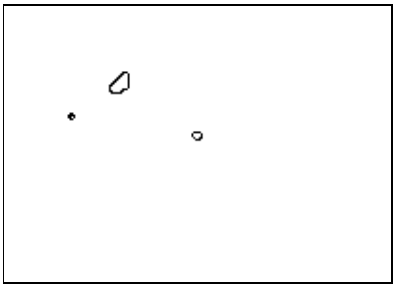

Figura 5.75: Esqueleto “barbeado”

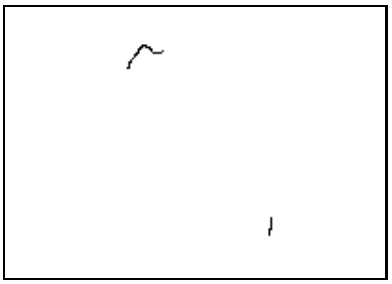

Figura 5.76: Resíduo

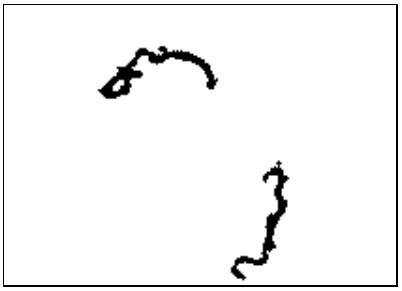

Figura 5.77: Filarioses

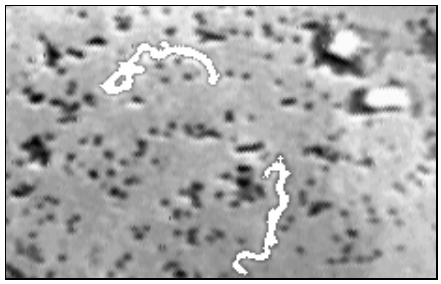

Figura 5.78: Imagem composta

5.4.2 Tecido Muscular

A imagem da fig. 5.79 mostra uma visão microscópica do corte transversal de um tecido muscular. O problema a ser resolvido é achar um operador que segmente as bordas das fibras que aparecem na imagem, tanto das fibras claras como das escuras. As principais dificuldades do problema são relativas à proximidade das fibras e à qualidade da imagem.

Como estamos interessados nas bordas dos objetos, vamos usar o Paradigma de Beucher. Para isso, devemos achar marcadores internos e externos para cada objeto na imagem. É fácil ver que os objetos escuros são facilmente marcáveis, por outro lado, para alguns objetos claros é difícil encontrar um marcador, ou mesmo distinguir suas bordas. Isso deve-se, em parte, à baixa resolução da imagem para o tamanho dos objetos.

Para tornar mais homogêneo os níveis de cinza de cada objeto na imagem, vamos aplicar um filtro composto por uma dilatação por um disco de diâmetro 10 e um fechamento por reconstrução por um quadrado elementar.

A imagem da fig. 5.80 mostra o resultado deste filtro sobre a imagem original. Note que o interior dos objetos está mais suave do que na imagem original e, também, que o filtro não alterou a posição das bordas da imagem. Isso é um bom motivo para usarmos esta imagem filtrada e não a imagem original, para achar as bordas. É bom lembrar que nem sempre isto é válido pois os filtros podem desaparecer com contornos importantes, casos nos quais o uso da imagem filtrada para achar contornos pode resultar em segmentações ruins.

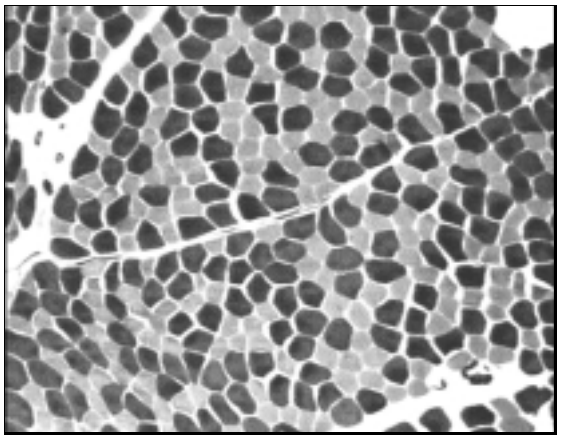

Figura 5.79: Tecido Muscular

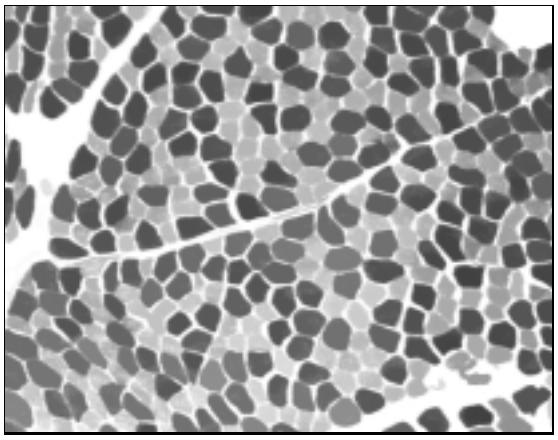

Figura 5.80: Primeira Filtragem

Os marcadores para as fibras mais escuras da imagem são achados tomando-se o máximo local da função distância aplicada à uma limiarização sobre uma filtragem da imagem original, que segmenta as fibras mais escuras das demais, como no exemplo 5.1.2.

Os marcadores para as fibras mais claras são achados tomando-se o máximo local da função distância aplicada a uma segunda filtragem (uma dilatação por um disco euclideano de diâmetro 15 seguido de uma erosão por um disco euclideano de diâmetro 8) sobre a imagem original (veja

fig. 5.81). O objetivo desta segunda filtragem é suavizar as fibras mais claras. A imagem fica bem deteriorada em relação à original, mas isso não afeta muito o encontro dos marcadores, inclusive porque eles são tratados para haver apenas um marcador por objeto.

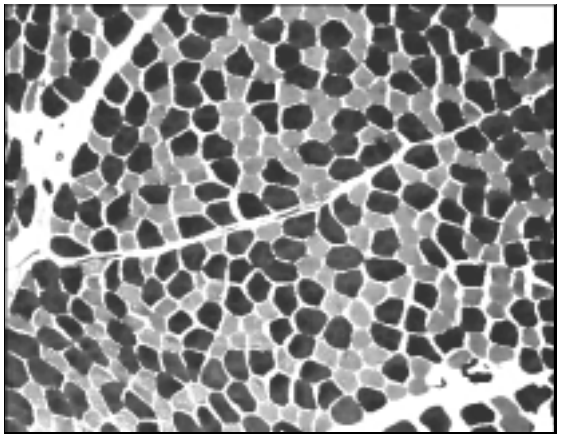

Figura 5.81: Segunda Filtragem

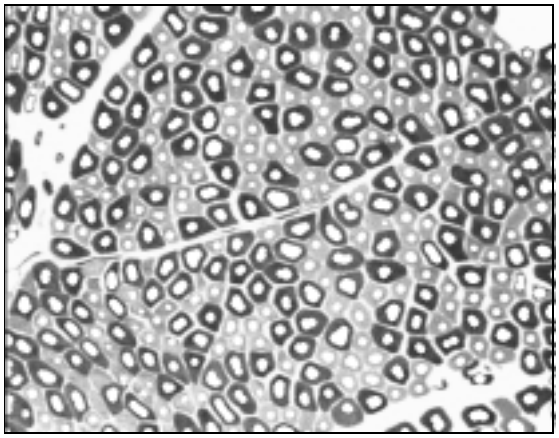

Figura 5.82: Marcadores

Devido à proximidade das fibras, não é necessário construir marcadores externos. Isso é uma pequena modificação do paradigma, necessária para processar esta imagem. A conseqüência principal será que as linhas de partição de águas encontradas pelo operador LPE não resultarão diretamente sobre as bordas dos objetos marcados.

A imagem da fig. 5.82 mostra o resultado do supremo das imagens dos marcadores com a imagem original e a imagem da fig. 5.83 mostra as linhas de partição encontradas pelo operador LPE após a mudança de homotopia do gradiente da imagem da fig. 5.80 usando os marcadores mostrados na imagem da fig. 5.82. Note que as bordas de muitos objetos já foram encontradas, mas o problema ainda não foi resolvido.

Para resolvê-lo vamos usar a imagem fig. 5.81 para ajudar no encontro das bordas dos objetos que ainda não estão corretas. A imagem da fig. 5.84 mostra o resultado de uma limiarização aplicada na imagem da fig. 5.81.

O número de fibras que se tocam nessa imagem é muito grande, para segmentá-las usaremos as linhas de partição encontradas anteriormente. A imagem da fig. 5.85 mostra o ínfimo entre a imagem da fig. 5.84 e a imagem da fig. 5.83.

Neste ponto, a maioria dos objetos já estão separados, embora tenham muitos buracos e estejam muito juntos. Para resolver este problema aplicamos o operador fecha-buracos e uma abertura por um disco de diâmetro cinco. O efeito desta abertura é alisar o contorno das fibras. A imagem da fig. 5.86 mostra o resultado da aplicação desta composição.

As bordas dos objetos desta imagem podem ser achados facilmente fazendo-se a subtração da imagem da fig. 5.86 pela sua própria erosão. A imagem da fig. 5.87 mostra o resultado final.

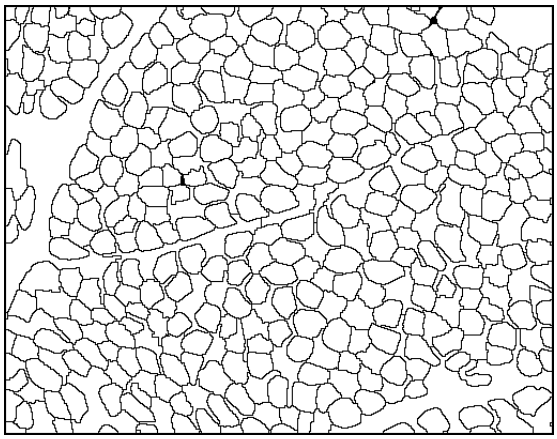

Figura 5.83: Linhas de Partição

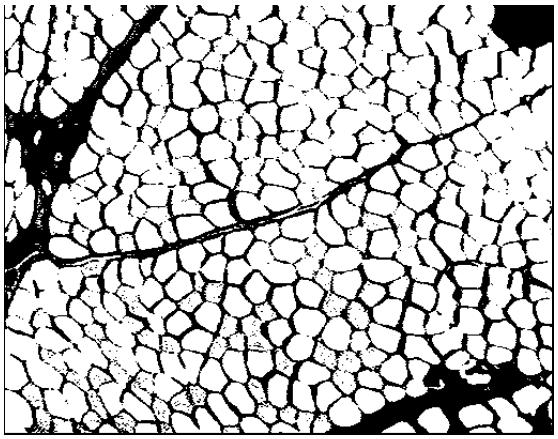

Figura 5.84: Limiarização

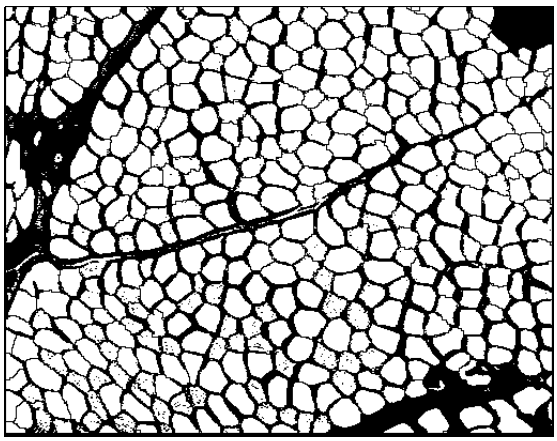

Figura 5.85: Ínfimo

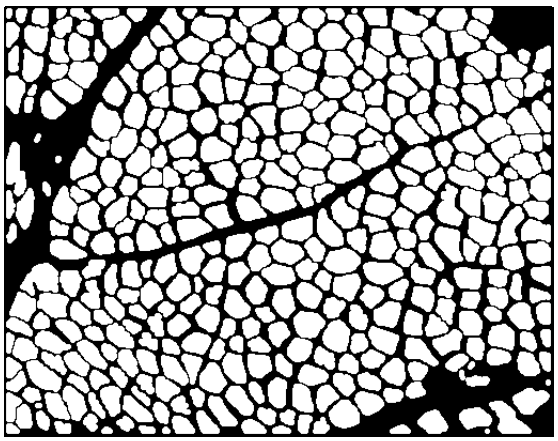

Figura 5.86: Objetos separados

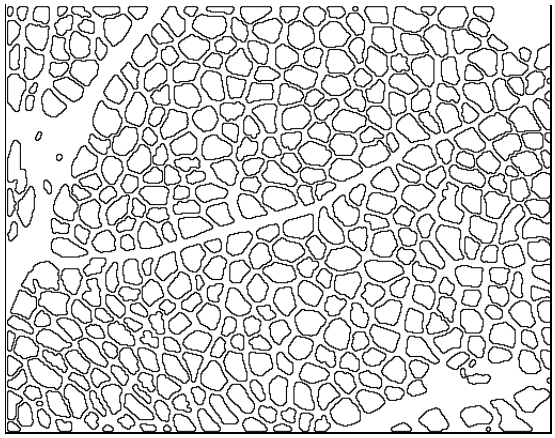

Figura 5.87: Resultado

Capítulo 6

Algoritmos "Morfológicos" rápidos

De acordo com a arquitetura de uma **MMach** [BB94], os operadores morfológicos são implementados por algoritmos (no caso de uma **MMach** implementada por "software") ou por circuitos (no caso de uma **MMach** implementada por "hardware") que realizam as operações e os operadores elementares.

Muitos operadores morfológicos realizam um grande número de iterações dos operadores elementares que os compõe para produzir o resultado final. Por exemplo, o operador de reconstrução morfológica é a iteração até a estabilidade do operador de dilatação condicional (que por sua vez é a composição do operador de dilatação com a operação de ínfimo). A cada iteração dos operadores elementares todo o domínio da função $f \in K^{\mathbf{E}}$, i.e., todos os pontos da imagem são processados. Assim, se um operador precisa de um número de iterações grande de operações elementares, o processamento de uma única imagem pode tornar-se lento, já que a dimensão típica de uma imagem é da ordem de milhares de pontos. No entanto, verifica-se que, em muitos casos, apenas alguns poucos pontos da imagem são realmente modificados a cada iteração, o que indica a possibilidade da construção de algoritmos mais eficientes.

Na literatura encontramos vários algoritmos rápidos projetados para computadores convencionais [1], normalmente apresentados como "Algoritmos Morfológicos" [VB89, Vin90, VS91, Vin93a], que imitam o funcionamento de operadores morfológicos complexos (frases da $\mathcal{ML}$) eficientemente. Estudaremos neste capítulo algumas técnicas para a construção desses algoritmos rápidos e como elas podem ser aplicadas para a construção de dois importantes algoritmos que imitam operadores importantes para segmentação de imagens, o algoritmo de reconstrução [Vin93a, Vin93b] e o algoritmo de LPE [VB89, Vin90, VS91].

Veremos que a eficiência dos algoritmos que serão estudados está, normalmente, baseada no uso de estruturas de dados convenientes, em geral filas, para armazenar ponteiros para os pontos que serão efetivamente modificados, ou ainda, ponteiros para os pontos que podem influenciar na modificação de pontos vizinhos a eles. Assim, a cada iteração, apenas os pontos que estão sendo apontados, ou seus vizinhos, é que serão processados, aumentando a eficiência do algoritmo.

A qualidade de um algoritmo "morfológico", no entanto, não é medida apenas pela sua veloci-

[1] Máquina RAM com um processador e cujas instruções são executadas uma a uma seqüencialmente.

dade. Há outras duas preocupações a considerar: precisão e flexibilidade [Vin93a]. Por precisão deve-se entender a capacidade do algoritmo de produzir o resultado mais próximo possível do resultado teórico do operador morfológico, o que é desejável, mas nem sempre é possível sem uma relativa perda de eficiência. [KR89, Vin93a]. Por flexibilidade deve-se entender que o algoritmo seja adaptável a outras grades e não apenas àquela para a qual ele foi implementado; que haja possibilidade da adaptação para várias métricas; e que possa ser iterado diversas vezes facilmente [LM84].

Veremos, também, como é difícil construir um algoritmo que possua todas as qualidades citadas. Uma das razões disso é simples, se queremos um algoritmo rápido, então não podemos gastar tempo para controlar a precisão. Outra razão é que, para conseguirmos um algoritmo eficiente, temos de restringí-lo normalmente à arquiteturas especializadas, o que diminui a sua flexibilidade.

6.1 Notações Básicas

Seja $\mathbf{E}$ um subconjunto retangular de $\mathbb{Z} \times \mathbb{Z}$ e $f \in K^{\mathbf{E}}$ uma imagem. A estrutura de dados mais conveniente para representar f é um vetor de pontos. A imagem da fig. 6.1 mostra um vetor de 35 posições e a imagem da fig. 6.2 o mesmo vetor disposto em forma de uma matriz. O valor de f em cada ponto de $\mathbf{E}$ será armazenado neste vetor de acordo com a ordem de varredura "**raster**" dos pontos de $\mathbf{E}$, i.e., visitando os pontos da esquerda para a direita e de cima para baixo (fig. 6.3).

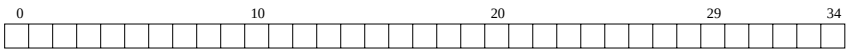

Figura 6.1: Representação Vetorial

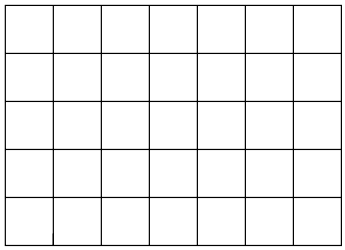

Figura 6.2: Representação Matricial

Existem outras estruturas de dados que podem ser usadas para representar uma imagem mas, como explicado em [Vin90, Vin93a], o vetor de pontos é a estrutura de dados melhor adaptada para a implementação de transformações morfológicas.

Os algoritmos que serão estudados neste capítulo serão apresentados em pseudo-linguagem de programação, semelhante ao **C**, e foram implementados em **ANSI C**.

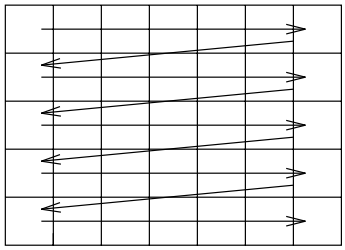

Figura 6.3: Varredura "raster"

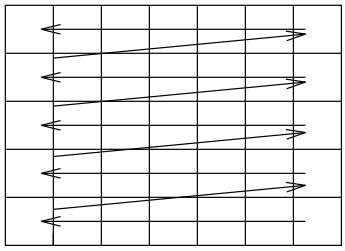

Figura 6.4: Varredura "anti-raster"

As convenções que iremos usar para apresentar os algoritmos têm sido amplamente empregadas em diversos artigos e livros, a menos de pequenas particularidades entre os diversos autores.

- A estrutura de grupo de instruções, indicada em **C** com "{" para o início, e "}" para o fim de um grupo, será indicada na pseudo-linguagem pela endentação das instruções, por exemplo, o corpo da instrução "**for**" no algoritmo da dilatação apresentado abaixo, na seção 6.2, é a linha 2.
- Os comandos de repetição "**for**", "**while**" e "**do while**", assim como os testes lógicos "**if-then-else**", têm as mesmas interpretações da linguagem **C**[2] .
- O sinal de ponto e vírgula "**;**" indica o fim de uma instrução.
- A flecha para a esquerda "$\leftarrow$" indica o comando de atribuição (o mesmo que "=" em **C**) e a instrução "$a \leftarrow b \leftarrow c$" indica uma atribuição múltipla.
- Os sinais "$=, \neq, <, >, \leq, \geq$" e as palavras "**and**" e "**or**" são usados na comparação de valores.
- Os sinais "$+, -$" indicam, respectivamente, as operações de soma, subtração e a palavra "**not**", a negação. Os sinais "++ e --" indicam as operações de incremento e decremento de variáveis, respectivamente.
- As palavras "**true**" e "**false**" serão usadas como valores lógicos para verdadeiro e falso, respectivamente. A palavra "**null**" será usada para indicar o valor vazio.

[2] Apesar da linguagem "**C**" não usar o comando "then", vamos usa-lo por questão de clareza.

- A função **return()** será usada como em **C**.
- O pseudo-comando "**Scan $\mathbb{E}$ in raster order (let p be the current pixel**[3]**)**" significa (fig. 6.3):

  "**for** $p = 0$ to $p =\mid \mathbb{E} \mid -1$ **do**",

  i.e., "para todos os pontos de $\mathbb{E}$, do primeiro ao último, faça".

  O pseudo-comando "**Scan $\mathbb{E}$ in anti-raster order (let p be the current pixel)**" é o comando de repetição na ordem contrária, i.e., "para todos os pontos de $\mathbb{E}$, do último para o primeiro (fig. 6.4).

  "**for** $p =\mid \mathbb{E} \mid -1$ to $p = 0$ **do**",

A notação $N_G(p)$ servirá para indicar o conjunto de todos os vizinhos do ponto p segundo a grade de conectividade G. Outras notações importantes são $N_G^+(p)$ e $N_G^-(p)$ que indicarão os conjuntos de todos os vizinhos do ponto p que são acessados antes de p na varredura "raster" e os vizinhos do ponto p que são acessados depois, respectivamente. As imagens da fig. 6.5 e fig. 6.6 mostram um exemplo dos conjuntos $N_G^+(p)$ e $N_G^-(p)$ quando a conectividade de G é 8. Note que o ponto p não pertence ao conjunto de seus vizinhos.

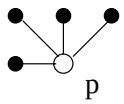

Figura 6.5: Vizinhos "raster"

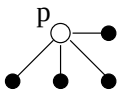

Figura 6.6: Vizinhos "anti-raster"

Em vários programas deste e do próximo capítulo usaremos uma estrutura de dados auxiliar do tipo *fila* (**FIFO** - "First In First Out") de pontos [Vin90, Vin93a].

Convencionaremos as seguintes operações básicas para usar esta estrutura de dados em nossos algoritmos.

- **queue_init**(W, n): aloca espaço para uma fila W com n pontos e a inicializa.
- **queue_add**(W, p): armazena um apontador para o ponto p na fila.
- **queue_first**(W): retorna o apontador para o ponto que está no início da fila e remove-o de lá.
- **queue_empty**(W): retorna verdadeiro se a fila está vazia e falso caso contrário.

[3]Em verdade, "current pixel coordinate", pois estamos tomando pontos do espaço $\mathbb{E}$. Estamos, também, usando a palavra "pixel" para designar apenas a coordenada do ponto, e não a coordenada mais o valor da função no ponto.

Dividimos nossa apresentação dos algoritmos que implementam operações entre imagens em 4 categorias, de acordo com a ordem que os pontos da imagem são transformados.

6.2 Algoritmos com Estrutura Paralela

Esta categoria de algoritmos é usada na implementação de todas as operações elementares da *MM*, por isso não podíamos deixar de mencioná-la.

Dizemos que um algoritmo tem estrutura paralela quando, qualquer que seja a ordem que os pontos sejam transformados, o resultado a cada etapa do algoritmo é o mesmo. Em outras palavras, a ordem que os pontos são transformados não altera o resultado.

Normalmente, estes algoritmos funcionam da seguinte maneira: seja $f \in K^{\mathbf{E}}$ a imagem a ser transformada. Para todos os pontos da imagem, independentemente da ordem, cada ponto é transformado de acordo com um certo critério, por exemplo, na dilatação cada ponto da imagem é avaliado em relação aos pontos pertencentes ao elemento estruturante centrado no ponto que está sendo transformado. O resultado da transformação será o maior valor encontrado dentro do elemento estruturante centrado naquele ponto e será armazenado em uma nova imagem.

Não importa a ordem que os pontos da imagem sejam dilatados o resultado será o mesmo, pois são avaliados apenas os pontos da imagem a ser transformada.

Algoritmo de dilatação

- Entrada: Imagem $f \in K^{\mathbf{E}}$
  Elemento estruturante $B \subset \mathbf{E}$
- Saída : Imagem dilatada $\delta_B(f) \in K^{\mathbf{E}}$

```
1       for all p ∈ E
2           δ_B(f)(p) = max{f(x) : x ∈ B + p};
3       return(f);
```

Algoritmos com estrutura paralela são implementados facilmente em arquiteturas paralelas. Uma forma de fazer isso é dividir o espaço $\mathbf{E}$ em tantas partições quanto o número de processadores disponíveis, distribuir o processamento de cada partição entre os processadores e aplicar o algoritmo apresentado anteriormente restrito à cada partição.

Outra característica interessante desses algoritmos é que eles são bastante flexíveis. É fácil ver que o algoritmo apresentado pode servir para qualquer tipo de grade e para espaços de dimensão qualquer.

Entretando, como já explicamos, flexibilidade, precisão e velocidade dificilmente podem ser conseguidas simultâneamente. Neste caso, o preço a ser pago é que os algoritmos com estrutura

paralela não são muito velozes quando implementados em arquiteturas convencionais.

6.3 Algoritmos com Estrutura seqüencial

Em arquiteturas convencionais, uma categoria de algoritmos muito utilizada para aumentar a eficiência do processamento entre imagens é a dos algoritmos com estrutura **seqüencial**. Dizemos que um algoritmo tem estrutura **seqüencial** quando:

1. Os pontos da imagem são processados um a um numa ordem pré-definida, em geral, na ordem de varredura "raster" ou "anti-raster" da imagem.
2. O valor de um ponto, computado usando o valor de seus vizinhos, é escrito na mesma imagem de entrada podendo, assim, ser usado para a computação do valor dos próximos pontos.

Dos itens 1 e 2 vemos que, ao contrário dos algoritmos com estrutura paralela, nos algoritmos com estrutura seqüencial a ordem é fundamental, já que o valor calculado de um ponto pode influenciar no valor de todos os pontos que serão processados posteriormente a ele.

Vamos dar um exemplo de um algoritmo com estrutura seqüencial [KR89] para o cálculo da Função Distância (dada uma imagem binária $f \in \{0,1\}^{\mathbf{E}}$ a função distância é um operador que atribui a cada ponto p de cada componente conexa da imagem a menor distância de p a um ponto x que não pertence àquela componente conexa, i.e., um ponto do "fundo" da imagem, veja a definição 3.3.27 no capítulo 3).

Algoritmo da Função Distância

- Entrada: Imagem binária $f \in \{0,1\}^{\mathbf{E}}$
- Saída : Imagem em níveis de cinza $f \in K^{\mathbf{E}}$

```
1    Scan E in raster order
2    (let p be the current pixel)
3        if(f(p) ≠ 0) then
4            f(p) ← inf{f(x) : x ∈ N_G^+(p) + 1};
5    Scan E in anti-raster order
6    (let p be the current pixel)
7        if(f(p) ≠ 0) then
8            f(p) ← inf{f(x) : x ∈ N_G^-(p) + 1};
9    return(f);
```

6.4 Algoritmos com Estrutura FIFO

A terceira categoria de algoritmos que gostaríamos de apresentar são os algoritmos baseados no uso de estruturas de dados do tipo FIFO ("Fisrt In First Out").

Nestes algoritmos os pontos são armazenados em uma fila e são processados na ordem em que estão armazenados. Esta ordem é extremamente dependente da imagem que está sendo processada. O resultado do processamento pode ser armazenado na própria imagem de entrada, ou numa cópia dela. Desta forma, os pontos já transformados podem, ou não, ser usados na transformação dos próximos pontos.

Para exemplificar uma implementação deste tipo, vamos mostrar o algoritmo que implementa a rotulação das componentes conexas de uma imagem binária. O operador de rotulação, definido no capítulo 3, atribui um valor diferente para cada componente conexa de uma imagem binária.

Este algoritmo foi sugerido por Jean Serra e está implementado na toolbox **MMach** sob a plataforma do **Khoros**.

Algoritmo de rotulação

- Entrada: Imagem binária $f \in \{0,1\}^{\mathbf{E}}$
- Saída : Imagem em níveis de cinza $g \in K^{\mathbf{E}}$

```
inteiro Label; /* número de componentes conexas */
inteiro longo Max; /* um número grande */
fila W; /* estrutura do tipo FIFO */

1     queue_init(W, |E|);
2     Label ← 1;
3     for all p ∈ E
4         if(f(p) = 1) then g(p) = Max;
5     Scan E in raster order
6     (let p be the current pixel)
7         if(g(p) = Max) then
8             queue_add(W, q);
9             g(p) ← Label;
10        while(queue_empty(W) = false)
11            q ← queue_first(W);
12            for every r ∈ N_G(q)
13                if (g(r) = Max) then
14                    fifo_add(W, p);
15                    g(q) ← Label;
16        Label++;
17    return(g);
```

O algoritmo de rotulação tem duas partes importantes. Na primeira parte (linhas 5 a 8) a imagem é percorrida no sentido "raster" até o encontro de uma componente conexa ainda não rotulada. O primeiro ponto não rotulado encontrado é rotulado e armazenado em uma fila.

Na segunda parte do algoritmo (linhas 9 a 13) é feita a rotulação de todos os pontos da componente num processo semelhante ao de busca em largura para grafos ou árvores [CLR90]. Os pontos vizinhos ao primeiro ponto encontrado são rotulados e armazenados, e assim por diante.

Após todos os pontos da componente conexa terem sido rotulados, a varredura é reinicializada a partir do ponto onde havia sido parada.

Dentre os algoritmos de rotulação conhecidos para computadores convencionais, este é o mais veloz. Além disso, ele é exato e pode ser adaptado para outras grades e dimensões.

Vamos apresentar duas formas eficientes de implementar o algoritmo de reconstrução usando as técnicas já apresentadas e, em seguida, vamos apresentar a quarta categoria de algoritmos, os algoritmos **mistos** ou **híbridos**, que mesclam o uso de técnicas seqüenciais e técnicas de filas para aumentar a eficiência.

6.5 Algoritmo de Reconstrução seqüencial

Como vimos no capítulo 3, o operador de reconstrução $\rho_{B,g}(f)$ é a iteração até a estabilidade de uma dilatação seguida de um ínfimo. É fácil ver que a implementação desse operador utilizando os algoritmos paralelos que implementam os operadores e operações elementares é muito ineficiente pois a cada iteração todos os pontos da imagem são examinados.

Vamos apresentar em seguida um algoritmo que pode ser usado para reconstrução tanto de imagens binárias como de imagens em níveis de cinza e que é mais eficiente que a implementação clássica.

Algoritmo de reconstrução seqüencial

- Entrada: Imagem marcadora $f \in K^{\mathbf{E}}$
  Imagem máscara $g \in K^{\mathbf{E}}$, $g \geq f$
- Saída : Imagem $f \in K^{\mathbf{E}}$

```
1     do
2         Scan E in raster order
3         (let p be the current pixel)
4             f(p) ← max{f(x) : x ∈ N_G^+(p) ∪ {p}} ∧ g(p);
6         Scan E in anti-raster order
7         (let p be the current pixel)
8             f(p) ← max{f(x) : x ∈ N_G^-(p) ∪ {p}} ∧ g(p);
9     while (not reached stability)
10    return(f);
```

A idéia deste algoritmo é propagar a informação para "baixo" quando da varredura "raster" e para "cima", quando da varredura "anti-raster". Este processo é repetido até a estabilidade, i.e., até que nenhum ponto tenha tido seu valor modificado após uma varredura completa ("raster" e "anti-raster").

Normalmente são necessárias poucas varreduras para reconstruir a imagem, a menos que a imagem tenha "estruturas espiraladas" como na imagem da fig. 6.7. Nestes casos são necessárias muitas varreduras e poucos pontos são modificados a cada varredura.

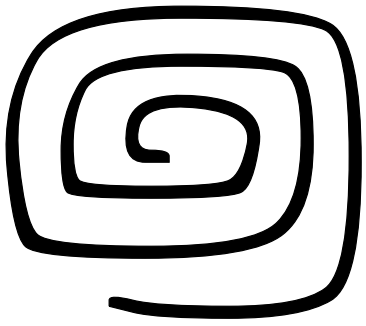

Figura 6.7: Componente espiralada

Como a imagem tem dimensões finitas e o valor da imagem máscara não ultrapassa o maior valor do conjunto K, é fácil ver que o algoritmo pára.

6.6 Algoritmo de filas para a reconstrução binária

Quando as imagens a serem processadas são binárias e o elemento estruturante é é um subconjunto de $\mathbb{B}$, é simples melhorar a eficiência do algoritmo usando estruturas do tipo FIFO.

Algoritmo de reconstrução binária

- Entrada: Imagem marcadora binária $f \in [0,1]^{\mathbf{E}}$
  Imagem máscara birária $g \in [0,1]^{\mathbf{E}}$, $g \geq f$
- Saída : Imagem binária $f \in [0,1]^{\mathbf{E}}$

fila W; /* estrutura do tipo FIFO */

```
1     queue_init(W, |E|);
2     Scan E in raster order
3     (let p be the current pixel)
4         if((f(p) = 1) & (∃q ∈ N_G(p) such that f(q) = 0)) then
5             queue_add(W, q);
6     while(queue_empty(W) = false)
7         p ← queue_first(W);
8         for every q ∈ N_G(p)
9             if((f(q) = 0) & (g(q) = 1)) then
10                f(q) ← 1;
11                queue_add(W, q);
12    return(f);
```

Este algoritmo foi proposto por Luc Vincent [Vin93a] e está baseado no uso de filas. A idéia é bastante simples, inicialmente (linhas 2 a 5) armazenam-se os pontos da borda de cada marcador (componente conexa) da imagem marcadora f.

Em seguida, as componentes conexas marcadas da imagem máscara serão reconstruídas na imagem marcadora a partir das bordas dos marcadores. Isso é feito nas linhas 6 a 11 onde, para cada ponto p armazenado na fila, examinam-se cada um de seus vizinhos e atualizam-se os valores desses vizinhos, se na imagem máscara os pontos que estão na mesma posição têm valor diferente de zero, ou não. Caso o valor de um ponto seja atualizado, ele é armazenado na fila (linha 11).

Em [Vin93a], Vincent propõe uma modificação do algoritmo apresentado para imagens em níveis de cinza. A idéia é fazer a reconstrução da imagem a partir dos máximos regionais da imagem. Isso é possível pois pode-se demonstrar que reconstruir uma imagem em níveis de cinza a partir da imagem máscara ou a partir das bordas dos máximos regionais da imagem máscara é equivalente.

6.7 Algoritmo Híbrido de Reconstrução

Damos o nome de híbrido a um tipo de algoritmo que usa técnicas vistas nos tipos seqüencial e por filas. O algoritmo que mostraremos a seguir foi inventado por Luc Vincent [Vin93a] e é o algoritmo mais rápido que encontramos na literatura para a implementação da reconstrução e da reconstrução dual em computadores convencionais.

O uso da estrutura híbrida justifica-se por causa que ambos os métodos apresentados anteriormente têm problemas. Por exemplo, o algoritmo de reconstrução por filas funciona bem para imagens binárias mas quando a idéia é aplicada para imagens em níveis de cinza, sua performance não é boa. O algoritmo de reconstrução seqüencial funciona bem se a imagem não possui estruturas encaracoladas, como explicado anteriormente.

Este algoritmo funciona bem para imagens binárias e em níveis de cinza, e pode ser implementado facilmente para diversos tipos de grades de conectividade mudando as relações de vizinhança N_G. Além disso, a adaptação do algoritmo para imagens n-dimensionais, $n > 2$ também é simples, bastando estabelecer a ordem de varredura da imagem.

Algoritmo híbrido de reconstrução

- Entrada: Imagem marcadora $f \in K^{\mathbf{E}}$
  Imagem máscara $g \in K^{\mathbf{E}}$, $g \leq f$
- Saída : Imagem $f \in K^{\mathbf{E}}$

```
1    Scan E in raster order
2    (let p be the current pixel)
3        f(p) ← max{f(x) : x ∈ N_G^+(p) ∪ {p}} ∧ g(p);
4    Scan E in anti-raster order
5    (let p be the current pixel)
6        f(p) ← max{f(x) : x ∈ N_G^-(p) ∪ {p}} ∧ g(p);
7        if(∃q ∈ N_G^-(p) such that (f(q) < f(p)) & (f(q) < g(q))) then
8            queue_add(W, q);
9    while(queue_empty(W) = false)
10       p ← queue_first(W);
11       for every q ∈ N_G(p)
12           if((f(q) < f(p)) & (f(q) ≠ g(q))) then
13               f(q) ← min{f(p), g(q)};
14               queue_add(W, q);
15   return(f);
```

A idéia do autor foi aplicar os métodos na ordem correta para que o algoritmo ficasse eficiente, i.e., primeiro é aplicado o método seqüencial, que propaga a informação rapidamente por toda a imagem. Para isso é aplicado uma varredura "raster" e uma "anti-raster" (linhas 1 a 6), sendo que durante a varredura "anti-raster" uma fila é construida para armazenar os pontos que ainda

têm chance de propagar as informações (linha 7). Como, em geral, são necessárias poucas varreduras para reconstruir a imagem (algumas vezes apenas uma é suficiente para imagens binárias!), fazendo apenas uma varredura completa ("raster" mais "anti-raster") e usando o método de filas para tratar os pontos da imagem que ainda não foram reconstruídos, o algoritmo torna-se muito eficiente.

6.8 Algoritmo de "Watershed"

O algoritmo mais eficiente que encontramos na literatura para implementar o operador LPE foi criado por Luc Vincent e Pierre Soille [Vin90, Vin91, Vin93a] e, desde então, tornou-se uma das ferramentas mais importantes de segmentação de imagens.

Existiam outras implementações anteriormente a essa, tanto no âmbito da topografia digital como no âmbito de **PDI**, mas elas eram ineficientes, ou de difícil adaptação a outras grades de conectividade, ou imprecisos. Tudo isso tornava difícil sua aplicação prática. Os primeiros a propor um algoritmo de LPE baseado em imersão foram Beucher e Lantuéjol, mas o algoritmo deles também era ineficiente para arquiteturas não especializadas [Vin91].

Apresentamos a metodologia de aplicação do operador LPE no capítulo 4 e vários exemplos práticos no capítulo 5. Vamos apresentar duas versões do algoritmo criado por Vincent e Soile. Ambas as versões usam a abordagem por **imersão**, apresentada no capítulo 3. Nelas Vincent utilizou estruturas FIFO em uma das partes do algoritmo e, também, uma técnica de ordenação por acesso direto de endereços ("sorting by address calculation") [IS56] para melhorar a eficiência dos algoritmos.

Vamos dividir o algoritmo em duas partes principais, a parte da ordenação e a parte da imersão.

6.8.1 Ordenação

A primeira parte do algoritmo ordena os pontos (endereços dos pontos) de acordo com o valor do nível de cinza de cada um deles. Os endereços são armazenados em um vetor de forma ordenada devido ao auxílio de um outro vetor que armazena a quantidade de pontos da imagem com um certo nível de cinza. Estando ordenados, dado um certo nível, os pontos com aquele nível de cinza podem ser endereçados diretamente.

A ordenação dos pontos tem duas etapas, na primeira é calculado o histograma dos níveis de cinza da imagem. O algoritmo que faz isso recebe uma imagem f e retorna um vetor de K posições (número de níveis de cinza possíveis na imagem), onde cada posição i do vetor armazena o número de vezes que o nível de cinza i ocorreu na imagem f.

Algoritmo para o cálculo do histograma da imagem - hist()

```
• Entrada: Imagem f ∈ K^E
• Saída :  Vetor H de |K| posições
           inteiro hmin; /* menor nível de cinza da imagem */
           inteiro hmax; /* maior nível de cinza da imagem */

1     H[ ] ← 0; /* Inicializa o vetor H */
2     hmin ← hmax ← f(0); /* Inicializa hmin e hmax */
3     for all p ∈ E
4         H[f(p)]++; /* Computa o histograma */
5         if( f(p) < hmin ) hmin ← f(p);
6         if( f(p) > hmax ) hmax ← f(p);
7     return(H,hmin,hmax);
```

Computado o histograma dos níveis de cinza da imagem, é possível construir um vetor ordenado de pontos, onde cada posição do vetor contém o endereço de um ponto na imagem. Isto é simples pois a informação de quantos pontos possuem um determinado nível de cinza é dada pelo histograma. A partir desta informação, um outro vetor, conhecido como histograma cumulativo é construído. Cada posição i do vetor armazena a quantidade de pontos da função f que possuem níveis de cinza menores do que i. Desta forma, para ordenar os pontos, i.e., organizar o endereço de cada ponto de acordo com o seu nível de cinza, basta percorrer a imagem novamente. Isso é feito pelo algoritmo apresentado a seguir, também conhecido como "Algoritmo para o cálculo da Tridistributiva".

Algoritmo para o cálculo da Tridistributiva - tridist()

```
• Entrada: Imagem f ∈ K^E
           inteiro hmin; /* menor nível de cinza da imagem */
           inteiro hmax; /* maior nível de cinza da imagem */
• Saída :  Vetor ims de |E| posições

vetor HC; /* vetor de k + 1 posições para armazenar o */
      /* histograma cumulativo da imagem */
1     HC ← 0;
2     for (i = hmin+1; i ≤ hmax +1; i++)
3         HC[i] ← HC[i − 1] + H[i − 1]; /* histograma cumulativo */
4     for all p ∈ E
5         ims[HC[f(p)]] ← p; /* coloca o endereço de p em ims */
6         HC[f(p)] ++;
7     return(ims);
```

6.8.2 Imersão

A segunda parte do algoritmo simula o processo de imersão usado no encontro das linhas de partição d'águas. Conforme a superfície vai sendo imersa, os pontos correspondentes a uma bacia de captação recebem um rótulo único para diferenciar as diversas bacias e os pontos correspondentes ao encontro de duas bacias de captação diferentes, i.e., cujos pontos possuem rótulos diferentes, recebem um rótulo especial que chamaremos ("**wshed**").

O algoritmo funciona da seguinte maneira, suponha que a superfície (imagem) já tenha sido imersa até um certo nível h, i.e., todos os pontos com nível de cinza menor ou igual a h já estejam rotulados. Como os pontos estão ordenados de acordo com os seus níveis de cinza, temos acesso direto a todos os pontos p que têm nível de cinza $h+1$ e podemos continuar a imersão para este nível dando a eles um certo valor temporário que chamaremos de "**mask**" (linhas 9 a 11). Aqueles pontos que têm pelo menos um vizinho já rotulado (seja um ponto pertencente a uma bacia de captação ou a uma linha de partição) recebem um novo valor que chamaremos de "**inqueue**" e são armazenados em uma fila (linhas 12 a 15).

Como explicamos anteriormente, o uso de filas melhora a eficiência dos algoritmos nos casos em que o valor de alguns poucos pontos da imagem são mudados por vez. Este é o caso da próxima parte do algoritmo (linhas 14 a 28), em que usa-se a fila construida anteriormente para fazer uma busca por largura na imagem e rotular todos os pontos do nível $h+1$, de acordo com a definição do operador LPE apresentado no capítulo 3. Esta rotulação é feita computando-se as zonas de influência geodésica (SKIZ) das bacias já rotuladas dentro das zonas que possuem os pontos marcados com o valor "**mask**" (linhas 14 a 28). Assim, o tamanho das bacias vai aumentando à medida em que os pontos marcados com "**mask**" vão recebendo o rótulo da bacia à qual eles são vizinhos (linhas 20 e 21). Se não usássemos filas, teríamos de fazer diversas varreduras na imagem toda para computar o SKIZ do nível h no nível $h+1$.

Os pontos que estiverem na fronteira entre duas bacias fazem parte das linhas de divisão d'águas da imagem e portanto recebem o rótulo "**wshed**" (linhas 22 e 23). Pode acontecer que rotulemos um ponto erradamente com o valor "**wshed**", mas isso é consertado nas linhas 20 e 21 do algoritmo. Isso acontece pois um ponto "**mask**" vizinho a um ponto "**wshed**" pode também ser um ponto "**wshed**" (linhas 24 a 26). A variável "**flag**" é usada neste caso para especificar se o valor "**wshed**" de um ponto p é devido a ele ser vizinho de um ponto cujo valor é "**wshed**" ou se é devido a ele estar na fronteira entre duas regiões com rótulos diferentes. Caso não usássemos essa variável, as linhas de partição poderiam tornar-se muito grossas, como no caso da imagem mostrada na fig. 6.8, onde as componentes são pontuais e estão representadas por um ponto preto, enquando as linhas estão representadas por um ponto cinza um pouco maior.

Falta ainda tratar o caso em que um novo mínimo (uma nova bacia) foi encontrado no nível $h+1$ e que não estava contida em nenhuma das bacias já existentes. Esse tratamento e rotulação dos pontos é feito nas linhas 29 a 39.

6.8.3 Análise de Tempo

O algoritmo de LPE é linear em relação ao número de pontos da imagem. Isto é fácil de ver pois:

- O algoritmo para o cálculo do histograma varre uma vez a imagem e uma vez o vetor do histograma. Como $|K| << |\mathbb{E}|$, esse último item é desprezível, assim podemos dizer que esta parte é linear no número de pontos da imagem.

- O algoritmo para a computação do histograma cumulativo e para a ordenação dos pontos segundo esse histograma varre uma vez a imagem e duas vezes o histograma cumulativo. Novamente, como $|K| << |\mathbf{E}|$, esta parte também é linear no número de pontos da imagem.

- O algoritmo que simula a imersão da superfície varre a imagem três vezes em média. Uma vez na inicialização de g com o valor "**init**" (linha 1); outra vez na expansão das bacias, onde cada ponto pode ser visitado até duas vezes; e mais uma vez na busca por novas bacias.

Como todas as etapas do algoritmo são lineares no número de pontos da imagem, o algoritmo todo também o é. Em [Vin91], os autores fazem uma comparação da segunda versão do algoritmo, que apresentaremos no final do capítulo, com diversas outras implementações.

6.8.4 Análise de Espaço

Um dos pontos fracos deste algoritmo é quanto à utilização de memória.

- São necessários $3 \times |\mathbb{E}|$ "bytes"[4]

- Para armazenar o vetor de endereços ordenados dos pontos da imagem são necessários $4 \times |\mathbf{E}|$ "bytes", em arquiteturas 32 bits.

- Para a fila são necessários $|\mathbf{E}|/3 \times 4$ "bytes", pois esse número é, em geral, suficiente para armazenar os endereços dos pontos na maioria das aplicações práticas.

- O espaço gasto para armazenar os vetores dos histogramas é desprezível em relação ao número de pontos da imagem.

Assim, o espaço aproximado que será necessário para processar uma imagem é de $8 \times |\mathbf{E}|$ "bytes". O algoritmo é linear em espaço também, mas a constante é alta.

[4] Usamos, em geral, imagens com 256 níveis de cinza para a imagem de entrada e imagens com 65536 níveis de cinza para a saída, pois o número de bacias pode ser grande.

Algoritmo rápido de LPE - Primeira versão

- Entrada: Imagem $f \in K^{\mathbf{E}}$
- Saída : Imagem $g \in K^{\mathbf{E}}$

Constante **mask** $\leftarrow$ -2; /* valor inicial de uma limiarização */
Constante **wshed** $\leftarrow$ 0; /* val. dos pontos que pertencem ao "watershed" */
Constante **init** $\leftarrow$ -1; /* val. in. dos pontos de g */
Constante **inqueue** $\leftarrow$ -3; /* val. para os pontos enfileirados */

```
1       for all p ∈ E  g(p) ← init; /* inicializa g */
2       rótulo_atual ← 0;
3       flag ← false;
4       hist(); /* computa o histograma dos níveis de cinza da imagem */
5       tridist(); /* computa o vetor ordenado de endereços */
6       for (h = hmin; h ≤ hmax; h++)
7       /* SKIZ geodésico do nível h − 1 dentro do nível h */
8           if (h ≠ 0) then start = HC[h − 1]; else start = 0;
9           for (i ← start; i < HC[h]; i++)
10              p = ims[i];
11              g(p) ← mask;
```

```
12              for every r ∈ N_G(p)
13                  if ((g(r) > 0) or (g(r) = wshed)) then
14                      queue_add(W, r);
15                      g(p) ← inqueue;
16          while (queue_empty(W) = false)
17              p ← queue_first(W);
18              for every r ∈ N_G(p)
19                  if (g(r) > 0) then
20                      if ((g(p) = inqueue or ((g(p) = wshed) and flag)) then
21                          g(p) ← g(r);
22                      else if ((g(p) > 0) and (g(p) ≠ g(r))) then
23                              g(p) ← wshed; flag ← false;
24                  else if (g(r) = wshed) then
25                              if (g(p) = inqueue) then
26                                  g(p) ← wshed; flag ← true;
27                  else if (g(r) = mask) then
28                              g(r) ← inqueue; queue_add(W, r);
29          for (j ← start; j < HC[i]; j++)
30              p = ims[j];
31              if (g(p) = mask) then
32                  g(p) ← ++rótulo_atual;
33                  queue_add(W, p);
34                  while (queue_empty(W) = false)
35                      r ← queue_first(W);
36                      for every q ∈ N_G(r)
37                          if (g(q) = mask) then
38                              queue_add(W, q);
39                              g(q) ← rótulo_atual;
40 return(g);
```

A motivação para fazer uma segunda versão do algoritmo é que, apesar do cuidado tomado introduzindo-se a variável "**flag**", as linhas de partição podem resultar erradas no caso em que haja dois mínimos próximos dentro de grandes platôs, como exemplificado na fig. 6.9.

Este problema é evitado na segunda versão que apresentaremos a seguir. A diferença entre as duas implementações é que, na segunda, usamos uma imagem de controle que armazena a distância geodésica e faz com que apenas os pontos que estão no meio do caminho entre duas bacias de captação pertençam às verdadeiras linhas de partição de águas (fig. 6.10).

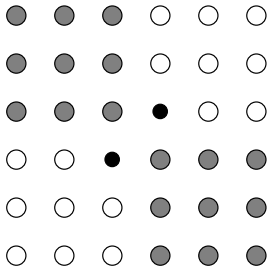

Figura 6.8: LPE muito grossa

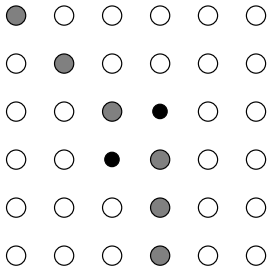

Figura 6.9: LPE mal posicionada

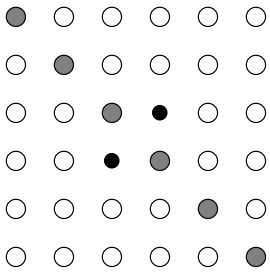

Figura 6.10: LPE correta

Algoritmo rápido de LPE - Segunda versão

- Entrada: Imagem $f \in K^{\mathbf{E}}$
- Saída : Imagem $g \in K^{\mathbf{E}}$

constante **mask** $\leftarrow$ -2; /* valor inicial de uma limiarização */
constante **wshed** $\leftarrow$ 0; /* val. dos pontos que pertencem ao "watershed" */
constante **init** $\leftarrow$ -1; /* val. in. dos pontos de g */
constante **inqueue** $\leftarrow$ -3; /* val. para os pontos enfileirados */

```
1       for all p ∈ E  g(p) ← init; /* inicializa g */
2       for all p ∈ E  dist(p) ← 0; /* inicializa dist(p) */
3       rótulo_atual ← 0;
4       hist(); /* computa o histograma dos níveis de cinza da imagem */
5       tridist(); /* computa o vetor ordenado de endereços */
6       for (h = hmin; h ≤ hmax; h++)
7       /* SKIZ geodésico do nível h − 1 dentro do nível h */
8           if (h ≠ 0) then start = HC[h − 1] else start = 0;
9           for (i ← start; i < HC[h]; i++)
10              p ← ims[i];
11              g(p) ← mask;
12              for every r ∈ N_G(p)
13                  if ((g(r) > 0) or (g(r) = wshed)) then
14                      queue_add(W, r); dist(p) ← 1;
15                      g(p) ← inqueue;
16          distância_atual ← 1;
17          queue_add(W,ponto_fictício);
18          do
19              p ← queue_first(W);
20              if (p = ponto_fictício) then
21                  if (queue_empty(W)) then BREAK
22                  else
23                      queue_add(W,ponto_fictício);
24                      distância_atual++;
25                      p ← queue_first(W);
```

```
26              for every r ∈ N_G(p)
27                  if ((dist(r) < distância_atual) and ((g(r) > 0) or (g(r) = wshed))) then
28                      if (g(r) > 0) then
29                          if ((g(p) = mask) or (g(p) = wshed)) then
30                              g(p) ← g(r);
31                          else if (g(p) ≠ g(r)) then g(p) ← wshed;
32                  else if (g(p) = mask) then g(p) ← wshed;
33                  else if ((g(r) = mask) and (dist(r) = 0)) then
34                          dist(r) ← distância_atual;
35                          queue_add(W, r);
36          while (true) /* indefinitely */
37          for (j ← start; j < HC[i]; j++)
38              p ← ims[j];
39              if (g(p) = mask) then
40                  g(p) ← ++rótulo_atual;
41                  queue_add(W, p);
42                  while (queue_empty(W) = false)
43                      r ← queue_first(W);
44                      for every q ∈ N_G(r)
45                          if (g(q) = mask) then
46                              queue_add(W, q);
47                              g(q) ← rótulo_atual;
48 return(g);
```

Capítulo 7

Decomposição de Operadores

No capítulo 3 vimos que qualquer operador definido sobre o reticulado das funções $K^{\mathbf{E}}$ com valores em $K^{\mathbf{E}}$ pode ser descrito pela linguagem morfológica $\mathcal{ML}$. Também vimos que uma frase da $\mathcal{ML}$ (um operador) pode ser implementada como um programa de uma Máquina Morfológica (MMach). Por outro lado, vimos no capítulo anterior a existência de algoritmos rápidos que imitam o funcionamento de operadores morfológicos sem serem programas de uma MMach. Dizemos que esses algoritmos "imitam" o funcionamento dos operadores e não "implementam" os operadores pois a estrutura deles não reflete a estrutura da MMach, i.e., eles não são construídos utilizando-se as operações e os operadores elementares da *MM* e, portanto, não são programas da MMach.

O desenvolvimento de algoritmos rápidos é válido e necessário pois a computação do mesmo resultado utilizando programas de uma MMach pode ser muito ineficiente. Entretanto, a principal desvantagem dessa abordagem é que não se pode aproveitar os algoritmos para outra coisa senão para aquilo que eles foram construídos. Por exemplo, se quiséssemos fazer um operador que computa eficientemente n-dilatações condicionais, não poderíamos aproveitar o algoritmo rápido de reconstrução [Vin93b]; teríamos que fazer um outro programa, apesar dele ter a dilatação condicional em sua estrutura.

Para evitar que para cada operador morfológico tenhamos de construir um programa eficiente, mostraremos que é possível escrevê-los como programas da MMach, com performance equivalente, se utilizarmos estruturas de dados convenientes (e algoritmos eficientes para tratar essas estruturas) para implementar os operadores e as operações elementares.

A estratégia para isto será de encontrar os operadores elementares "escondidos" nos algoritmos rápidos e aproveitar as idéias e as estruturas de dados que esses algoritmos utilizam para implementar operadores elementares rápidos para as MMachs.

7.1 Notações Básicas

No capítulo 2 definimos a fronteira de uma imagem binária como sendo o conjunto de todos os pontos que tenham pelo menos um vizinho pertencente ao fundo da imagem.

Vamos ampliar a noção de fronteira para imagens binárias ou em níveis de cinza, definindo a noção de fronteira de influência de um elemento estruturante restrito[1] $B \subset \mathbb{B}, o \in B$.

Definição 7.1.1 *A fronteira de influência do elemento estruturante B na imagem $f \in K^{\mathbf{E}}$ é o subconjunto dado por*

$$\partial f = \{x \in \mathbb{E} : \exists p \in B + x, f(p) < f(x)\}$$

Como pode-se notar, se a imagem tem poucos platôs, é possível que a maior parte dos pontos da imagem pertença à fronteira.

7.2 Representações Equivalentes de Algoritmos com Estrutura FIFO

Veremos nesta seção como os operadores elementares podem ser representados de forma equivalente à representada no capítulo 3, de modo que fiquem mais eficientes quando implementados por algoritmos com estrutura FIFO.

7.2.1 Dilatação e dilatação condicional

Para computar a dilatação, como definida no capítulo 3, temos de computá-la para todos os pontos de $\mathbb{E}$ e, usualmente, apenas alguns pontos terão seus valores modificados. Isto pode ser visto na fig. 7.1, onde é mostrada a representação numérica de uma imagem em níveis de cinza e sua dilatação por um losango, considerando uma grade com conectividade 4.

Os pontos que foram modificados pela dilatação, a cada operação, são os pontos 4-vizinhos dos pontos marcados com colchetes (pontos da fronteira ∂f_B). No algoritmo rápido de reconstrução por estruturas FIFO, vimos como uma seqüência de dilatações condicionais pode ser implementada eficientemente. Usando esta idéia, vamos definir e mostrar que a dilatação pode ser computada sabendo-se apenas quais são os pontos da fronteira ∂f_B. Esta idéia foi usada também por [VV88] e por [d'O92] para fazer algoritmos rápidos para dilatações e erosões binárias.

Definição 7.2.1 *Seja $f \in K^{\mathbf{E}}$, a dilatação da função f pelo elemento estruturante B é definida por:*

$$\delta_B(f)(x) = \begin{cases} \bigvee\limits_{y \in B + x \cap \partial f} f(y), & \text{se } x \in \partial f \oplus B \cap \mathbf{E} \\ f(x) & \text{caso contrário} \end{cases}$$

[1] Os operadores que apresentaremos podem funcionar sem a primeira restrição para o elemento estruturante, mas existe perda de eficiência nesses casos. Desta forma, daqui em diante, sempre que escrevermos B, estaremos falando do elemento estruturante restrito.

0	0	0	0	0	0	0
0	[1]	[1]	[1]	[1]	0	0
0	[1]	[2]	1	1	[1]	0
0	[1]	1	[2]	1	[1]	0
0	[1]	1	1	1	[1]	0
0	[1]	[1]	[1]	[1]	[1]	0
0	0	0	0	0	0	0

(a)

0	[1]	1	1	[1]	0	0
[1]	1	[2]	1	1	[1]	0
1	[2]	2	[2]	1	1	[1]
1	1	[2]	2	[2]	1	1
1	1	1	[2]	1	1	1
[1]	1	1	1	1	1	[1]
0	[1]	1	1	1	[1]	0

(b)

1	1	[2]	1	1	[1]	0
1	[2]	2	[2]	1	1	[1]
[2]	2	2	2	[2]	1	1
1	[2]	2	2	2	[2]	1
1	1	[2]	2	[2]	1	1
1	1	1	[2]	1	1	1
1	1	1	1	1	1	1

(c)

Figura 7.1: Imagem em níveis de cinza e dilatações por um losango

Pode-se mostrar que esta representação é equivalente à definição de dilatação apresentada no capítulo 3.

Demonstração: Por definição,

$$\delta_B(f)(x) = \bigvee_{y \in B^t \cap \mathbf{E}} f(x+y) = \bigvee_{y \in (B+x) \cap \mathbf{E}} f(y)$$

Vamos mostrar a equivalência por partes; primeiro mostraremos para os pontos que não pertencem à borda dilatada e depois para os pontos que pertencem à borda.

Se um ponto x é tal que $x \notin \partial f \oplus B$, não existe um ponto $y \in (B+x)$ tal que $f(y) > f(x)$, i.e., todo ponto $y \in (B+x)$ satisfaz $f(y) = f(x)$, e portanto, $\delta_B(f)(x) = \bigvee_{y \in (B+x) \cap \mathbf{E}} f(y) = f(x)$.

Se um ponto x é tal que $x \in \partial f \oplus B$, então $\exists y \in (B+x) \cap \partial f$ tal que $f(y) > f(x)$, portanto,

$$\delta_B(f)(x) = \bigvee_{y \in (B+x) \cap \mathbf{E}} f(y) = \bigvee_{y \in (B+x) \cap \partial f} f(y),$$

por causa que não existe $p \in (B+x) - \partial f$ tal que $f(p) > f(x)$, por definição de ∂f.

□

Note que se quisermos computar a segunda dilatação de f, podemos facilmente computar a nova fronteira $\partial f'$, onde f' é o resultado da primeira dilatação. Há duas maneiras de fazer isso, uma é computando a fronteira a partir de f' pela definição. A outra é mais simples e eficiente pois basta tomar os pontos $x \in B + p$, $p \in \partial f$ tais que $\exists q \in B + x$ e $f'(q) < f'(x)$ (veja fig. 7.1 b e fig. 7.1 c).

Este raciocínio pode ser usado recursivamente para o cômputo da n-dilatação de uma função f por um elemento estruturante B. Sejam $f = f_0$, $\partial f = \partial f_0$ e as seguintes relações:

$$\begin{aligned} f_{i+1}(x) &= \delta_B(f_i) \\ \partial f_{i+1} &= \{x \in \partial f_i \oplus B \cap \mathbf{E} : \exists p \in B + x, f_{i+1}(p) < f_{i+1}(x)\} \end{aligned}$$

Definição 7.2.2 *A n-dilatação de $f \in K^{\mathbf{E}}$ por um elemento estruturante B é dada por $\delta_B^n(f) = f_n$.*

Esta definição também é equivalente à definição de n-dilatação apresentada no capítulo 3 e sua demonstração é semelhante à anterior.

Para computar a dilatação condicional, n-dilatação condicional e a reconstrução pode-se usar o operador de dilatação definido para as estruturas FIFO e compô-lo para formar estes outros operadores, da mesma maneira como foram definidos no capítulo 3 . Entretanto, para otimizar o cômputo do operador dilatação condicional de uma função f condicionada à uma função máscara g, definiremos uma operação de ínfimo que opere apenas nos pontos que foram modificados pela dilatação. Se não fizéssemos isso, não haveria vantagem em usar a representação otimizada da dilatação.

Como já observamos, os pontos que são modificados pela dilatação estão contidos no conjunto $\partial f \oplus B \cap \mathbf{E}$, e são estes que terão que ser examinados pela operação de ínfimo. Partiremos da hipótese que $f \leq g$, como é o caso da dilatação condicional.

Definição 7.2.3 *Seja $f, g \in K^{\mathbf{E}}$ e $f \leq g$. O ínfimo entre a função $h = \delta_B(f)$ e a função máscara g é definida, para todo $x \in \mathbf{E}$,*

$$inf(h,g)(x) = \begin{cases} h(x) \bigwedge g(x), & \textit{se } x \in \partial f \oplus B \cap \mathbf{E} \\ h(x) & \textit{caso contrário} \end{cases}$$

É fácil ver que esta definição apenas otimiza o cômputo do ínfimo e é equivalente à definição da operação de ínfimo definida no capítulo 3.

No caso da reconstrução para imagens binárias, as estruturas FIFO tornam o método eficiente pois apenas os pontos das bordas das componentes conexas da imagem são armazenados na fila.

Para imagens em níveis de cinza, o método pode não funcionar tão bem pois a quantidade de pontos de borda pode ser muito grande, como já foi explicado no capítulo 6.

7.2.2 Erosão e Erosão Condicional

As mesmas idéias que foram usadas para criar uma representação equivalente para a dilatação serão usadas para criar uma representação equivalente para a erosão. Na imagem da fig. 7.2b mostramos o resultado do operador erosão sobre a imagem da fig. 7.2a por losango. Note que a erosão afeta apenas os pontos da fronteira ∂f_B e não os seus vizinhos, como no caso da dilatação; isso refletirá na sua representação equivalente. Note também que a definição de fronteira dada anteriormente é robusta e funciona tanto para a dilatação como para a erosão.

0	0	0	0	0	0	0
0	[1]	[1]	[1]	[1]	0	0
0	[1]	[2]	1	1	[1]	0
0	[1]	1	[2]	1	[1]	0
0	[1]	1	1	1	[1]	0
0	[1]	[1]	[1]	[1]	[1]	0
0	0	0	0	0	0	0

(a)

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	[1]	[1]	[1]	0	0
0	0	[1]	1	[1]	0	0
0	0	[1]	[1]	[1]	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

(b)

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	[1]	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

(c)

Figura 7.2: Imagem em níveis de cinza (a), sua primeira (b) e segunda erosão (c) por um losango.

Definição 7.2.4 *Seja $f \in K^{\mathbf{E}}$. A erosão da função f por um elemento estruturante B é dada por, para todo $x \in \mathbf{E}$,*

$$\varepsilon_B(f)(x) = \begin{cases} \bigwedge\limits_{y \in B+x} f(y), \text{ se } x \in \partial f \\ f(x) \quad \text{caso contrário} \end{cases}$$

Pode-se mostrar que esta representação é equivalente à definição de erosão apresentada no capítulo 3.

Se quisermos computar uma segunda erosão, basta computar a nova fronteira da imagem $\partial f'$, onde $f' = \varepsilon_B(f)$. Isto pode ser feito tomando-se os pontos $x \in B + p$, $p \in \partial f$, tal que $\exists q \in B + x$ e $f'(q) < f'(x)$ (veja fig. 7.2b). Somente os pontos da fronteira $\partial f'$ são erodidos (veja fig. 7.2c).

Desta forma, podemos escrever a fórmula para o cômputo de n-erosões de uma função f por um elemento estruturante B. A prova é semelhante à prova feita para a n-dilatação, i.e., por indução.

Seja $f = f_0$, $\partial f = \partial f_0$ e as seguintes relações:

$$\begin{aligned} f_{i+1} &= \varepsilon_B(f_i) \\ \partial f_{i+1} &= \{x \in \partial f_i \oplus B \cap \mathbf{E} : \exists p \in B + x, f_{i+1}(p) < f_{i+1}(x)\} \end{aligned}$$

Definição 7.2.5 *A n-erosão de $f \in K^{\mathbf{E}}$ por um elemento estruturante B é dada por, $\varepsilon_B^n(f) = f_n$.*

Esta definição também é equivalente à definição de n-erosão apresentada no capítulo 3.

Para computar a erosão condicional, a n-erosão condicional e a reconstrução dual pode-se usar o operador de erosão definido para as estruturas FIFO e compô-lo para formar estes outros operadores, da mesma maneira como foram definidos no capítulo 3. Entretanto, para otimizar o cômputo do operador erosão condicional de uma função f condicionada à uma função máscara g, definiremos uma operação de supremo que opere apenas nos pontos que foram modificados pela erosão; da mesma forma que definimos uma representação para a operação de ínfimo.

Como já observamos, os pontos que são modificados pela erosão estão contidos no conjunto $\partial f \cap \mathbf{E}$, e são estes que terão que ser examinados pela operação de supremo. Partiremos da hipótese que $f \geq g$, como é o caso da erosão condicional.

Definição 7.2.6 *Seja $f, g \in K^{\mathbf{E}}$ e $f \geq g$. O supremo entre uma função $h = \varepsilon_B(f)$ e uma função máscara g é definido por, para todo $x \in \mathbb{E}$:*

$$sup(h, g)(x) = \begin{cases} h(x) \bigvee g(x), & se\ x \in \partial f \cap \mathbb{E} \\ h(x), & caso\ contrário \end{cases}$$

É fácil ver que esta definição apenas otimiza o cômputo do supremo e é equivalente à definição da operação de supremo definida no capítulo 3 e, dessa forma, aqueles operadores podem ser computados eficientemente.

7.3 Representação Equivalente para Estruturas Seqüenciais

Como vimos no capítulo 6, algoritmos com estrutura seqüencial mudam o valor de apenas um ponto de cada vez, sendo que o valor de um ponto já visitado pode ser usado para modificar os pontos que serão visitados após ele.

Podemos construir um operador morfológico para fazer o mesmo, desde que usemos o conceito de funções estruturantes[2] Se tomarmos uma função estruturante b tal que:

[2] Uma função estruturante $b : \mathbb{E} \to P(\mathbb{E})$ associa a cada ponto p de $\mathbb{E}$ um elemento estruturante.

$$b_p(x) = \begin{cases} B, & \text{se } x = p \\ \mathrm{O} & \text{caso contrário} \end{cases}$$

então, para cada ponto $p \in \mathbf{E}$ haverá uma função estruturante que se usada pela dilatação ou erosão modificara apenas o valor do ponto p.

Compondo dilatações ou erosões para cada ponto $p \in \mathbf{E}$ e percorrendo a imagem na ordem "raster" ou "anti-raster", teremos o resultado equivalente ao de um algoritmo seqüencial.

Vamos aplicar essas idéias para escrever a reconstrução com estrutura seqüencial com uma frase da $\mathcal{ML}$.

7.3.1 Reconstrução

A representação equivalente para a reconstrução com estrutura seqüencial é baseada na definição de duas funções estruturantes, uma para representar a parte "raster" e outra para a parte "anti-raster" de cada iteração.

Definição 7.3.1 *Seja $B^+ \subset \mathbb{B}$ um elemento estruturante formado pelos pontos $x \in B$ que são acessados antes da origem O de $\mathbb{B}$ na varredura "raster". A função estruturante para o processo "raster", $b_p^+(x)$ é definida por, para todo $x \in \mathbf{E}$:*

$$b_p^+(x) = \begin{cases} B^+ + x & \text{se } x = p \\ \{x\} & \text{caso contrário} \end{cases}.$$

Pela definição acima, a função estruturante irá depender do ponto que estiver sendo modificado na imagem quando a varredura for na ordem "raster" e valerá B^+ nesse ponto e $\{x\}$ em qualquer outro ponto da imagem. Um operador que use essa função estruturante, por exemplo a dilatação, poderá modificar apenas o ponto p e não modificará os outros.

A definição equivalente para a função estruturante para a varredura "anti-raster" é dada a seguir.

Definição 7.3.2 *Seja $B^- \subset \mathbb{B}$ um elemento estruturante formado pelos pontos $x \in B$ que são acessados depois da origem O de $\mathbb{B}$ na varredura "raster". A função estruturante para o processo "anti-raster", $b_p^-(x)$ é definida por, para todo $x \in \mathbf{E}$:*

$$b_p^-(x) = \begin{cases} B^- + x & \text{se } x = p \\ \{x\} & \text{caso contrário} \end{cases}.$$

A parte "raster" é definida pela composição de dilatações condicionais pela função estruturante $b_p^+(x)$.

Definição 7.3.3 *Sejam $f, g \in K^{\mathbf{E}}$, $f(x) \leq g(x)$ e B um elemento estruturante. A fórmula definida por, para todo $x \in \mathbb{E}$,*

$$f_{p+1} = \delta_{b_p^+}(f_p) \wedge g$$

onde $f_1 = f$ e p é um índice para os pontos de $\mathbf{E}$ percorridos na ordem "raster". A iteração para todos os pontos de $\mathbf{E}$, será denotada $\Psi^+_{B,g}(f) = f_n$, representa, em uma frase da $\mathcal{ML}$, o processo "raster" do algoritmo do capítulo 6.

A parte "anti-raster" é definida de forma semelhante pela composição de dilatações condicionais pela função estruturante $b_p^-(x)$.

Definição 7.3.4 *Sejam $f, g \in K^{\mathbf{E}}$, $f(x) \leq g(x)$ e $B \subset \mathbb{B}$. A fórmula definida por, para todo $x \in \mathbf{E}$,*

$$f_{p+1} = \delta_{b_p^-}(f_p) \wedge g$$

onde $f_1 = f$ e p é um índice para os pontos de $\mathbf{E}$ percorridos na ordem "anti-raster". A iteração para todos os pontos de $\mathbb{E}$, será denotada $\Psi^-_{B,g}(f) = f_n$, representa, em uma frase da $\mathcal{ML}$, o processo "anti-raster" do algoritmo do capítulo 6.

Assim, como no caso das representações para estruturas FIFO, devemos adequar a representação da operação de ínfimo que é usada na dilatação condicional.

Definição 7.3.5 *Seja $f, g \in K^{\mathbf{E}}$. A operação definida por, para todo $x \in \mathbf{E}$,*

$$(f \wedge g)(x) = \begin{cases} f(x) \wedge g(x), & \text{se } x = p \\ f(x), & \text{caso contrário} \end{cases}$$

é equivalente à operação de ínfimo apresentada no capítulo 3 para este processo com estrutura seqüencial pois a cada passo a dilatação transforma apenas o ponto p.

Definição 7.3.6 *O operador definido por,*

$$\Psi_{B,g} = \Psi^-_{B,g} \circ \Psi^+_{B,g}$$

representa a composição das partes "raster" e "anti-raster" do algoritmo de reconstrução.

O operador de reconstrução é a composição da função $\Psi_{B,g}$ iterada até a estabilidade. A mesma idéia pode ser usada para a representação do operador de reconstrução dual.

7.4 Representação de Algoritmos com estruturas híbridas

Vimos até aqui como podemos representar os operadores com estrutura FIFO e com estrutura seqüencial. Para escrever um operador com estrutura híbrida numa frase da $\mathcal{ML}$, basta escrever o operador como uma composição dos operadores e operações já apresentados para essas estruturas.

7.4.1 Reconstrução

No capítulo 6 vimos que o algoritmo híbrido de reconstrução é experimentalmente muito mais eficiente do que os com estruturas puramente FIFO ou seqüencial. Desta forma, é importante que saibamos escrevê-los como uma frase da $\mathcal{ML}$ usando as representações equivalentes dos operadores elementares para aquelas estruturas.

Definição 7.4.1 *Seja $f, g \in K^{\mathbf{E}}$, $f \leq g$ e B um elemento estruturante. A representação da reconstrução híbrida de g por f é definida por, para todo $x \in \mathbb{E}$:*

$$\rho_{B,g}(\Psi_{B,g}(f)),$$

onde $\rho_{B,g}$ é a reconstrução por estrutura FIFO e $\Psi_{B,g}$ é a reconstrução por estrutura seqüencial.

Para que possa ser feita essa composição, temos que calcular a fronteira de influência $\partial\Psi_{B^+,g}(f)$ para o elemento estruturante B^+, i.e., temos que armazenar numa fila todos os pontos $p \in \mathbb{E}$ tais que:

$$\Psi_{B^+,g}(f)(p) > \Psi_{B^+,g}(f)(q) \quad \text{e} \quad \Psi_{B^+,g}(f)(q) < g(q),$$

assim,

$$\partial\Psi_{B^+,g}(f) = \{p \in \mathbf{E} : \exists q \in (B^+ + p), \Psi_{B^+,g}(f)(p) > \Psi_{B^+,g}(f)(q) \quad \text{e} \quad \Psi_{B^+,g}(f)(q) < g(q)\}$$

Note que esta definição é consistente com a previamente definida pois ela forma um subconjunto dela. É fácil ver que a inclusão dos outros pontos que estão fora desta definição não mudaria o resultado final.

7.5 Algoritmos para as Representações Equivalentes

As representações descritas nas seções anteriores são facilmente traduzidas para uma linguagem algorítmica em pseudo código.

7.5.1 Algoritmos para Representações Equivalentes com estrutura FIFO

Do mesmo modo que quando apresentamos os algoritmos rápidos, são usados um vetor de pontos para representar uma imagem e uma fila para armazenar os pontos que serão modificados ou que poderão modificar os seus vizinhos.

Fronteira de influência de B numa imagem

Há várias maneiras de inicializar a fronteira de influência de B numa imagem, uma delas é dada por um algoritmo específico que extrai as bordas das componentes conexas de uma imagem binária.

Procedure find_binary_border$(f, B, \partial f)$

- Input f - input image
  B - structuring element
- Output ∂f - queue with the border of f

Scan $\mathbb{E}$ in raster order
(let p be the current pixel)
  if $(\exists\ q \in B + p, f(p) = 1$ and $f(q) = 0)$
    fifo_add(p);

Um algoritmo menos específico, que serve tanto para imagens binárias como para imagens em níveis de cinza é o seguinte:

Procedure find_gray_border$(f, B, \partial f)$

- Input f - input image
  B - structuring element
- Output ∂f - queue with the border of f

Scan $\mathbb{E}$ in raster order
(let p be the current pixel)
  if $(\exists\ q \in B + p, f(q) < f(p))$
    fifo_add(p);

No caso dos algoritmos híbridos, precisamos de um algoritmo diferente, que também serve para imagens binárias ou em níveis de cinza.

Procedure hyb_border$(f, g, B, \partial f)$
- Input f - marker image
  g - mask image
  B - structuring element
- Output ∂f - queue with the border of f

Scan $\mathbb{E}$ in anti-raster order
(let p be the current pixel)

if ($\exists\ q \in B + p, f(q) < f(p)$ and $f(q) < g(q)$)
fifo_add(p);

Ínfimo

Pela definição 7.2.3 o ínfimo entre g e h, para $h = \delta_B(f)$ e $f \leq g$, pode ser computada pelo algoritmo abaixo.

Procedure infqueue($h, g, B, \partial f$)

- Input h - input image
  g - mask image
  B - structuring element
  ∂f - queue with the frontier of f
- Output h - input image

while (queue_empty(∂f) = false);
$p \leftarrow$ queue_first(∂f);
for every $q \in B + p$
if ($h(q) > g(q)$) $h(q) \leftarrow g(q)$;

O algoritmo para o supremo é similar ao apresentado acima.

Dilatação

Usando a definição 7.2.1, podemos facilmente escrever um algoritmo para a dilatação.

Procedure dilqueue($f, B, \partial f, \partial f'$)

- Input f - input image
  B - structuring element
  ∂f - queue with the frontier of f
- Output f - input image
  $\partial f'$ - queue for the next iteration

while (queue_empty(∂f) = false)
$p \leftarrow$ queue_first(∂f);
for every $q \in B + p$
if ($f(q) < f(p)$) $f(q) \leftarrow f(p)$;
if ($\exists r \in B + q, f(r) < f(q)$)

queue_add($\partial f', q$);

Reconstrução

Este algoritmo de reconstrução é similar à sua representação dada pela definição 7.3.6. A fronteira é inicializada pelo algoritmo find_binary_border() ou find_gray_border(), dependendo do tipo de imagem de entrada.

Procedure queuerec($f, g, B, \partial f$)
• Input f - original image
g - mask image
B - structuring element
∂f - queue with the frontier of f
• Output f - result image

while (queue_empty(∂f)=false)
dilqueue($f, B, \partial f, \partial f'$);
infqueue($f, g, \partial f$);
$\partial f \leftarrow \partial f'$;
$\partial f' \leftarrow$ null;

7.5.2 Algoritmos baseados em estruturas seqüenciais

A implementação destes algoritmos é muito fácil e a complexidade é constante para os operadores e operações básicas.

Algoritmo de ínfimo para estruturas seqüenciais

O algoritmo de ínfimo para estruturas seqüenciais é semelhante ao algoritmo de ínfimo usual, mas ele é pontual, como visto na definição 7.3.5.

Procedure inf_seq(f, g, i)
• Input f - original image
g - mask image
i - point that will have its value changed
• Output f - result image

$f(i) \leftarrow f(i) \wedge g(i)$;

O algoritmo do supremo é semelhante ao apresentado acima.

Reconstrução Seqüencial

Este algoritmo serve para implementar o operador definido em 7.3.6. Ele é similar à primeira parte do algoritmo descrito na seção 7.4.1. Uma seqüência de varreduras irão ser feitas até a estabilidade. Para determinar isso, ao final de cada varredura completa, "raster" mais "anti-raster", é verificado se a imagem foi modificada. Uma maneira mais eficiente é introduzir uma variável booleana de controle. Se um ponto é modificado durante uma varredura, muda-se o estado da variável. O algoritmo de reconstrução irá parar quando a variável booleana não tenha mudado de estado após uma varredura completa. O algoritmo termina quando, após uma varredura completa, a imagem não tiver sido modificada.

Procedure rec_seq(f, g, B)
- Input f - original image
  g - mask image
  B - structuring element
- Output f - result image

repeat until stability
  Scan **E** in raster order
  (let p be the current pixel)
    dil_seq(f, B^+, p);
    inf_seq(f, g, i);
  Scan **E** in anti-raster order
    (let p be the current pixel)
    dil_seq(f, B^-, p);
    inf_seq(f, g, i);

7.5.3 Algoritmos Híbridos

A motivação para construir algoritmos híbridos é balancear as vantagens e desvantagens de duas ou mais técnicas. No caso dos operadores que utilizam estruturas FIFO, a maior dificuldade é construir uma fila com uma quantidade de pontos pequena.

Reconstrução

Este algoritmo implementa o operador descrito em 7.4.1.

Procedure rec_hyb(f, g, B)
- Input f - original image

g - mask image
B - structuring element
• Output f - result image

Scan $\mathbf{E}$ in raster order
(let p be the current pixel)
dil_seq(f, B^+, p);
inf_seq(f, g, i);
Scan $\mathbf{E}$ in anti-raster order
(let p be the current pixel)
dil_seq(f, B^-, p);
inf_seq(f, g, i);
hybborder$(f, g, B^+, \partial f)$;
while (queue_empty(∂f)=false)
dilqueue$(f, B, \partial f, \partial f')$;
infqueue$(f, g, B, \partial f)$;
$\partial f \leftarrow \partial f'$;
$\partial f' \leftarrow$ null;

7.6 Resultados Experimentais

Nesta seção mostraremos os resultados que obtivemos nos testes para verificar a eficiência dos algoritmos.

Os algoritmos que apresentamos na seção anterior foram implementados em **ANSI C** e integrados ao **Khoros** para serem testados. Os testes foram realizados em um computador do tipo PC com processador Pentium 90 MHz e 40Mb de memória RAM.

Como os tempos que medimos são muito pequenos em comparação com a precisão do relógio do computador, repetimos cada experimento 10 vezes e tomamos a média dos tempos obtidos. Omitimos propositadamente a barra de erros dos gráficos por julgarmos que ela não adiciona informação importante para o entendimento e verificação dos resultados experimentais.

As imagens de entrada para todos os experimentos têm dimensões aproximadas de 512×512 pontos, i.e., todas elas têm acima de 250.000 pontos. O tempo para calcular a borda da imagem não foi considerado na tomada do tempo dos experimentos pois a borda inicial é calculada apenas uma vez para cada imagem.

7.6.1 Resultados para a Dilatação

Em cada experimento fizemos de 1 a 100 dilatações da imagem de entrada e medimos os tempos totais (i.e., o tempo que o processador demorou para calcular n-dilatações, $1 \leq n \leq 100$) e os tempos médios para cada uma das n dilatações.

Os algoritmos que analisamos foram:

- MMACH-BIT - é o algoritmo de dilatação implementado na MMACH para tratar imagens binárias em formato bit compactado, i.e., cada 8 pontos da imagem é representado em um byte e as operações "AND" e "OR" do processador são usadas para otimizar o processamento.
- MMACH - é o algoritmo de dilatação implementado na MMACH para imagens binárias ou em níveis de cinza em formato byte ou short.
- QUEUE - é o algoritmo de dilatação que usa a estrutura de fila, apresentado na seção anterior.

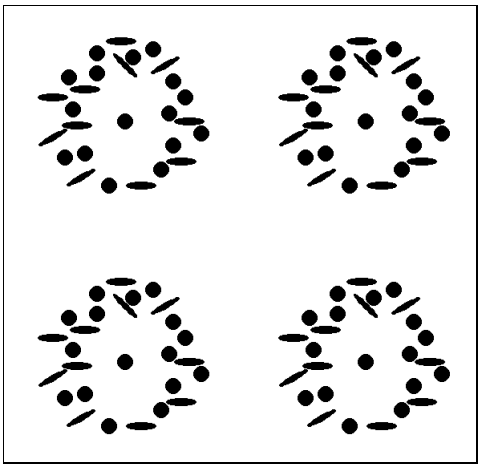

Figura 7.3: arc512

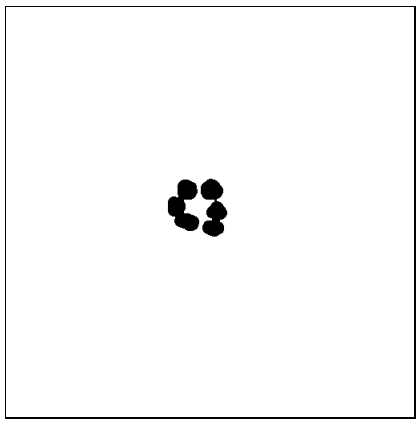

Figura 7.4: blobspq

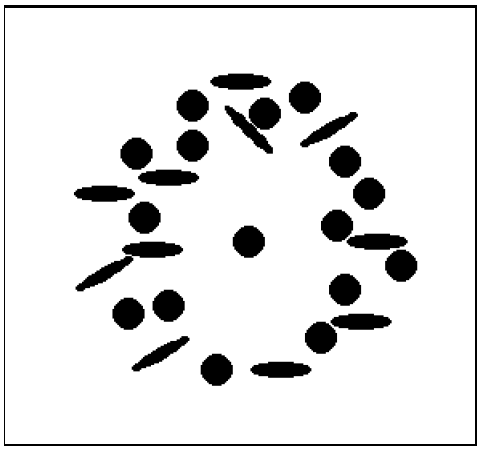

Figura 7.5: arc_exp

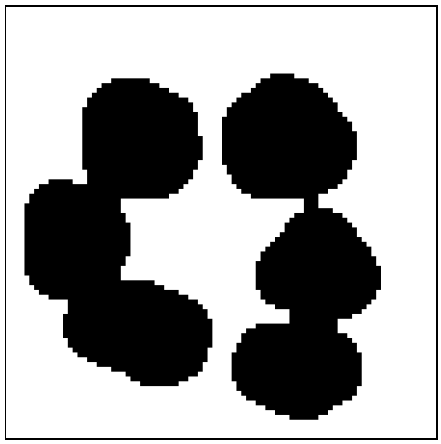

Figura 7.6: blobsgd

Figura 7.7: feath

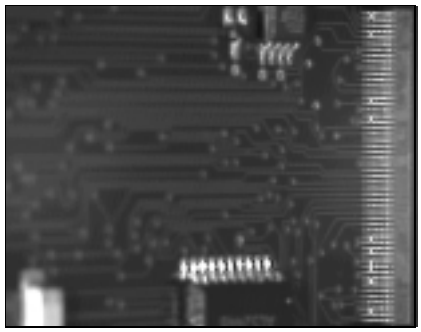

Figura 7.8: solda

Dilatação em Imagens Binárias

Os três primeiros experimentos foram realizados tendo uma imagem binária como entrada (fig. 7.3) e variando o elemento estruturante.

Experimento 1

Os gráficos da fig. 7.9 e fig. 7.10 mostram, respectivamente, os resultados obtidos para o tempo total de cada n-dilatação e para o tempo médio gasto em cada dilatação em uma n-dilatação, ambos para o experimento 1. Usamos um elemento estruturante do tipo linha horizontal.

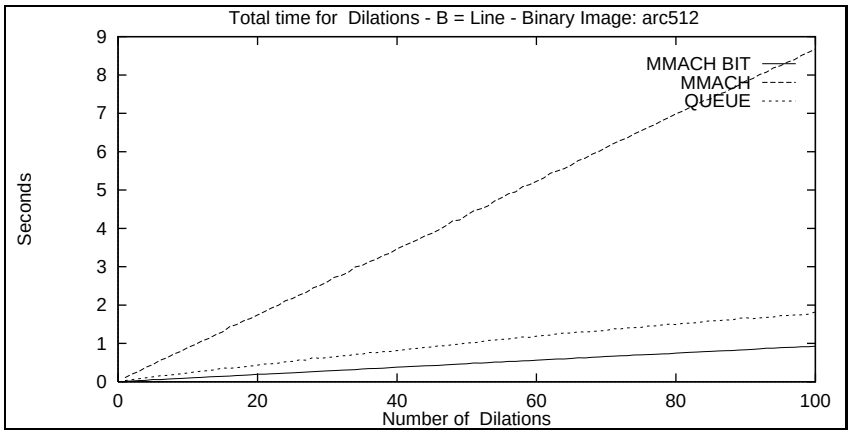

Figura 7.9: Tempo total da dilatação para a imagem arc512, B = linha

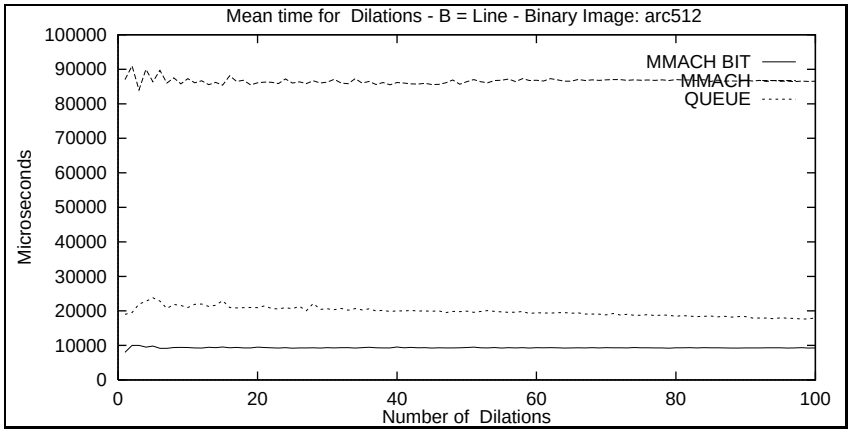

Figura 7.10: Tempo médio da dilatação para a imagem arc512, B = linha

Experimento 2

Este experimento é idêntico ao primeiro, a menos do elemento estruturante que agora é a cruz (ou losango). Os gráficos da fig. 7.11 e da fig. 7.12 mostram, respectivamente, os mesmos resultados para o experimento 2.

A diferença nos resultados em relação ao primeiro experimento é que o tempo total e o tempo médio aumentaram (pois o número de pontos vizinhos que têm que ser observados é dependente do elemento estruturante)

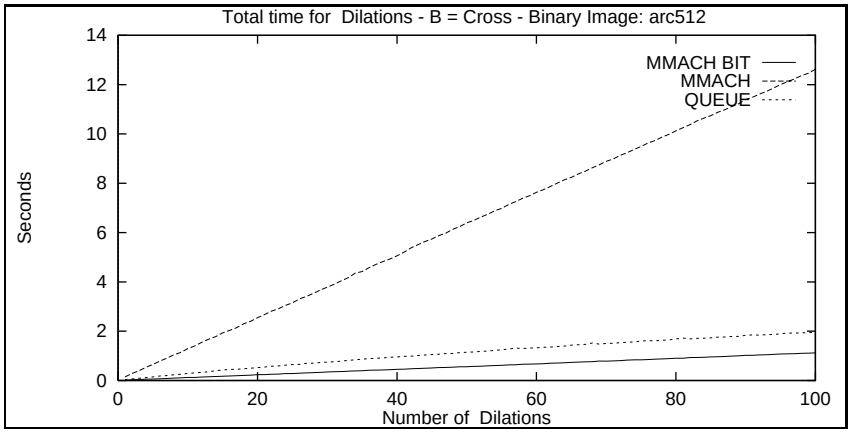

Figura 7.11: Tempo total da dilatação para a imagem arc512, B = cruz

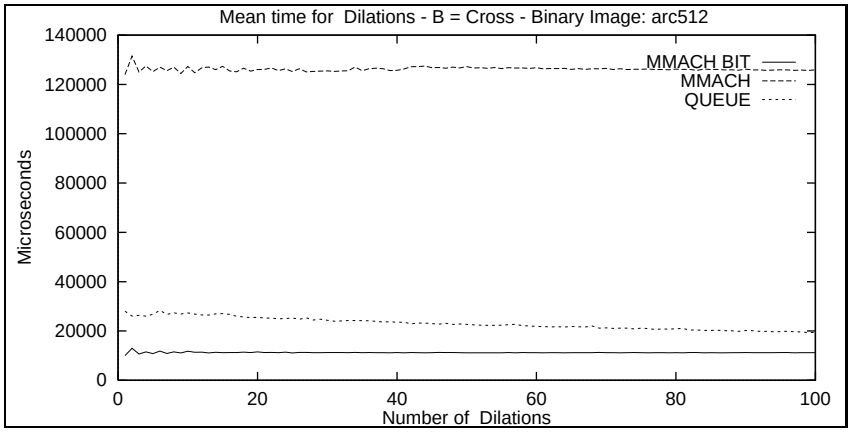

Figura 7.12: Tempo médio da dilatação para a imagem arc512, B = cruz

Experimento 3

Este experimento é idêntico ao primeiro, a menos do elemento estruturante que agora é o quadrado elementar. Os gráficos da fig. 7.13 e da fig. 7.14 mostram, respectivamente, os mesmos resultados para o experimento 3.

Neste experimento o tempo total e o tempo médio são, em média, maiores do que no experimento 2, como era de se esperar. A diferença principal é que a curva de tempo médio do algoritmo QUEUE tem uma queda ainda mais acentuada do que no experimento 2 e chega a ser menor do que o tempo médio do mesmo algoritmo no experimento 2 para $n \geq 91$.

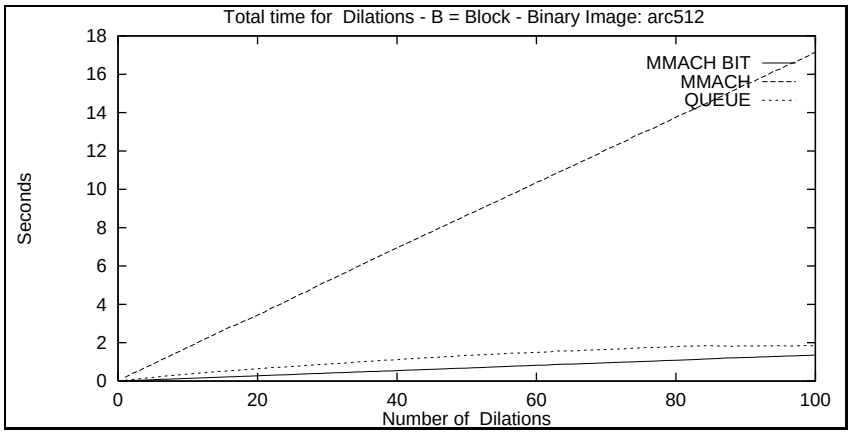

Figura 7.13: Tempo total da dilatação para a imagem arc512, B = quadrado elementar

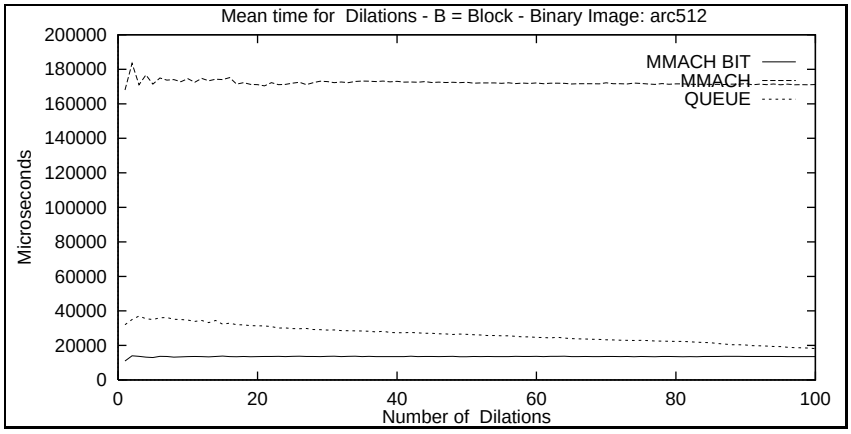

Figura 7.14: Tempo médio da dilatação para a imagem arc512, B = quadrado elementar

O próximo experimento é idêntico ao experimento 2 (onde foi usado o losango como elemento estruturante), a menos da imagem de entrada, que tem menos componentes conexas (veja fig. 7.4).

Experimento 4

Os gráficos da fig. 7.15 e da fig. 7.16 mostram, respectivamente, os resultados obtidos para o tempo total de cada n-dilatação e para o tempo médio gasto em cada dilatação em uma n-dilatação para o experimento 4.

Neste caso, como a imagem tem poucas componentes conexas, o tempo total e o tempo médio são menores do que experimento 2.

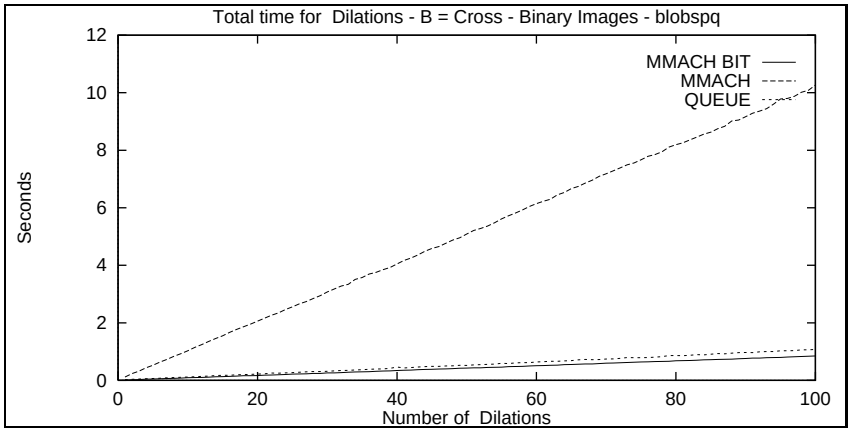

Figura 7.15: Tempo total da dilatação para a imagem blobspq, B = losango

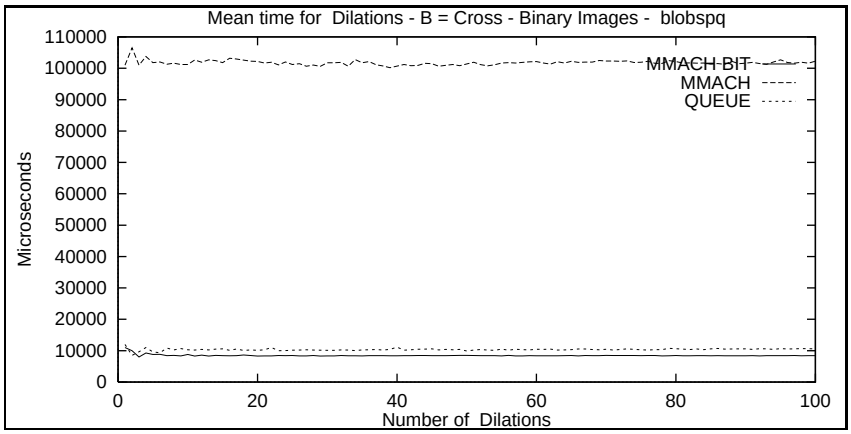

Figura 7.16: Tempo médio da dilatação para a imagem blobspq, B = losango

Em geral, nos gráficos que mostram o tempo total de uma n-dilatação (fig. 7.9, fig. 7.11, fig. 7.13 e fig. 7.15) vemos que o algoritmo mais rápido é o MMACH-BIT, como era de se

esperar pela especificidade da implementação. O algoritmo mais lento é o MMACH, pois ele tem que processar todos os pontos em formato byte. O algoritmo QUEUE tem um desempenho intermediário, mas mais parecido com o da MMACH-BIT.

Analisando os gráfico que mostram os tempos médios de cada dilatação em uma n-dilatação (fig. 7.10, fig. 7.12, fig. 7.14 e fig. 7.16), vemos que o tempo médio de cada dilatação nos algoritmos MMACH e MMACH-BIT é aproximadamente constante, o que é de se esperar pois cada dilatação é linear em relação ao número de pontos da imagem e esse número é constante. Já a curva do algoritmo QUEUE mostra uma pequena declinação média devida ao número de bordas irem diminuindo com o número de dilatações, o que é de se esperar já que o número de componentes (e consequentemente o número de bordas) vai diminuindo.

Dilatação em Imagens em Níveis de Cinza

Os próximos experimentos relativos a $n-$dilatações foram feitos para imagens em níveis de cinza. Nestes experimentos o algoritmo MMACH-BIT não foi considerado pois ele é específico para imagens binárias em formato bit compactado.

Os experimentos números 5, 6 e 7 tiveram como entrada a imagem da fig. 7.7, que é muito usada em artigos sobre processamento de imagens. Além disso, ela é uma imagem com poucos platôs[3], o que permite-nos testar a eficiência de nossos algoritmos sobre uma imagem complexa.

Experimento 5

Neste experimento foi usado o elemento estruturante horizontal, a "linha" como chamamos anteriormente. Os gráficos da fig. 7.17 e da fig. 7.18 mostram, respectivamente, os resultados obtidos para o tempo total de cada n-dilatação e para o tempo médio gasto em cada dilatação em uma n-dilatação para o experimento 5.

Como pode-se ver, o tempo de dilatação MMACH é melhor do que o do algoritmo QUEUE até a sexagésima sexta dilatação, depois disso, o número de pontos na borda é pequeno o suficiente para o algoritmo QUEUE ficar mais rápido do que o algoritmo MMACH. Este resultado era esperado pois o número de pontos inicial na fila é muito grande em relação ao número de pontos da imagem.

[3] Regiões conexas de pontos de $\mathbb{E}$ tais que $f(p)$ é constante para todos os pontos da região.

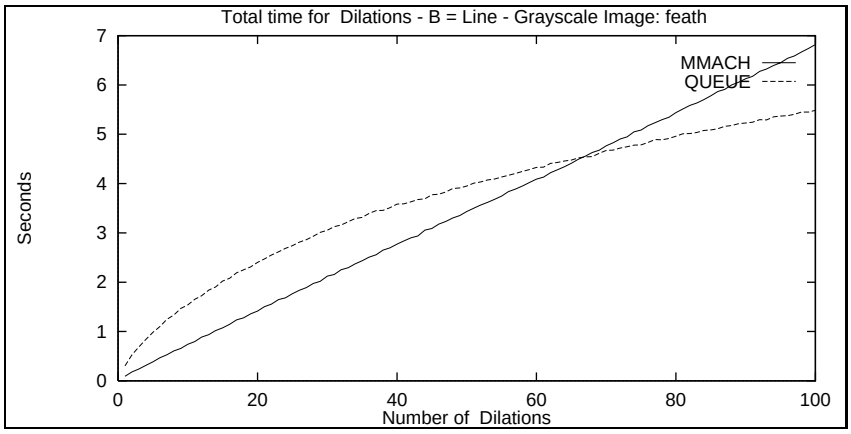

Figura 7.17: Tempo total da dilatação para a imagem feath, B = linha

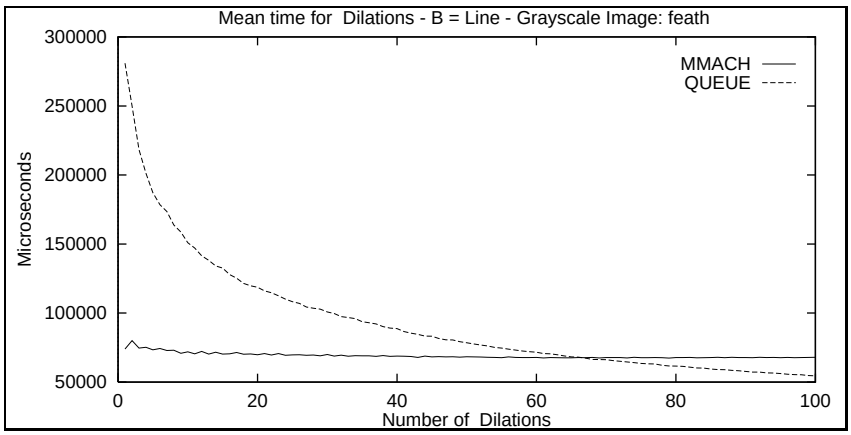

Figura 7.18: Tempo médio da dilatação para a imagem feath, B = linha

Experimento 6

Este experimento é idêntico ao experimento anterior, a menos que foi usado a cruz como elemento estruturante. Os gráficos da fig. 7.19 e da fig. 7.20 mostram, respectivamente, os mesmos resultados para o experimento 6.

Novamente o tempo de dilatação MMACH é melhor do que o do algoritmo QUEUE, mas desta vez ele foi melhor em todo o experimento. A partir de um certo número de dilatações o outro (QUEUE) ficará melhor, certamente, mas esse número é maior do que 100 dilatações.

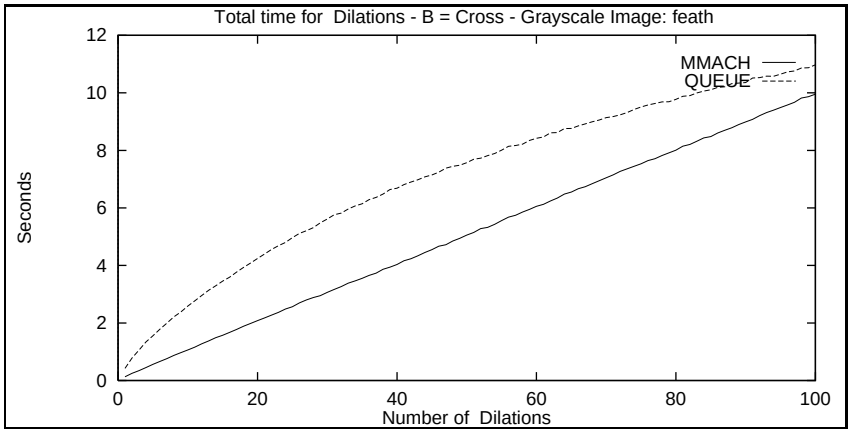

Figura 7.19: Tempo total da dilatação para a imagem feath, B = losango

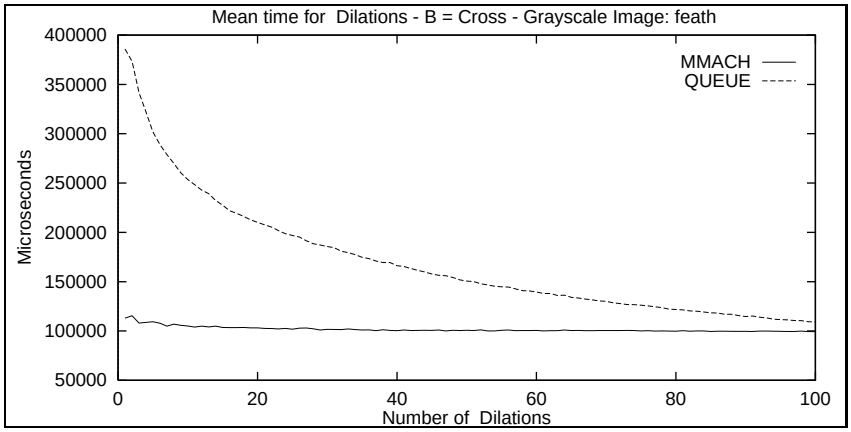

Figura 7.20: Tempo médio da dilatação para a imagem feath, B = losango

Experimento 7

Este experimento é idêntico ao experimento 5, a menos que foi usado o quadrado elementar como elemento estruturante. Os gráficos da fig. 7.21 e da fig. 7.22 mostram, respectivamente, os mesmos resultados para o experimento 6.

Desta vez, os tempos do algoritmo QUEUE ficam melhores a partir da octogésima sétima dilatação. Isso justifica-se pois, apesar do número de vizinhos a serem observados ser maior, os platôs aumentam mais rapidamente que no experimento anterior.

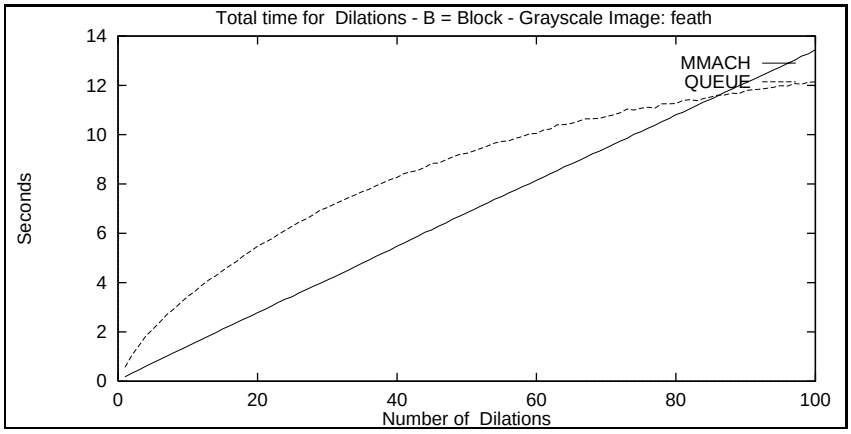

Figura 7.21: Tempo total da dilatação para a imagem feath, B = quadrado elementar

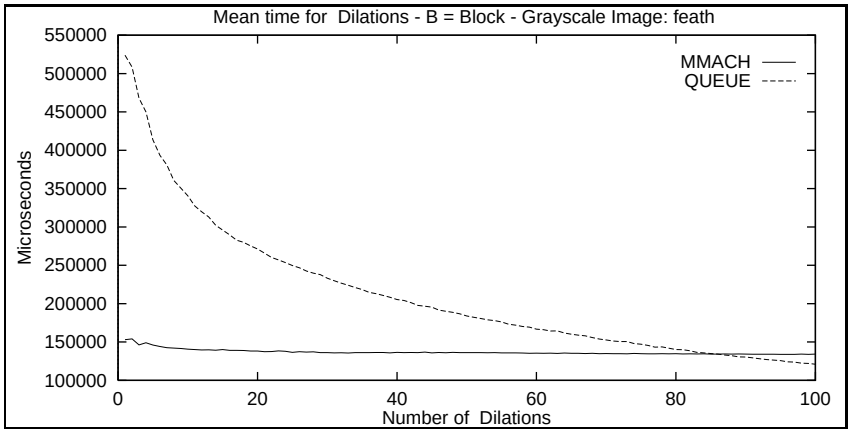

Figura 7.22: Tempo médio da dilatação para a imagem feath, B = quadrado elementar

O próximo experimento é idêntico ao experimento 6 (onde foi usado a cruz como elemento estruturante), a menos da imagem de entrada (veja fig. 7.8). Neste caso, a imagem é mais simples (pois tem menos contrastes) do que a imagem da fig. 7.7. Isso implica que o número de platôs deve ser maior e também que os platôs são maiores, i.e., têm área maior.

Experimento 8

Os gráficos da fig. 7.23 e da fig. 7.24 mostram, respectivamente, os resultados obtidos para o tempo total de cada n-dilatação e para o tempo médio gasto em cada dilatação em uma n-dilatação para o experimento 8.

Note que, neste caso, o algoritmo QUEUE já é mais rápido do que o algoritmo MMACH a partir da décima oitava dilatação.

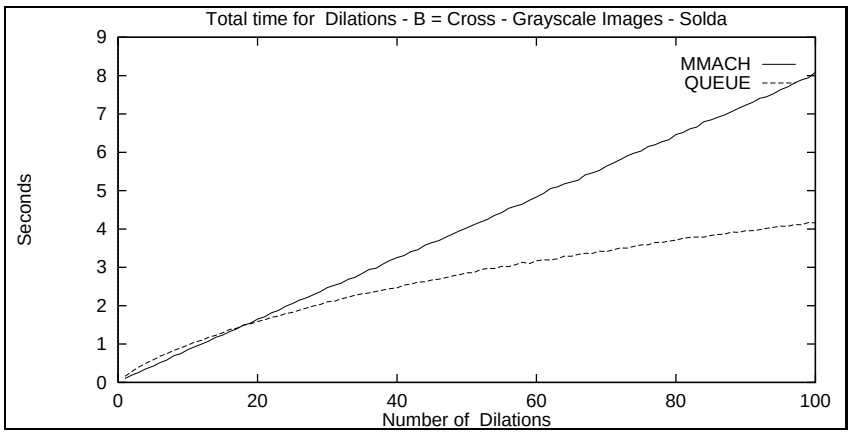

Figura 7.23: Tempo total da dilatação para a imagem solda, B = losango

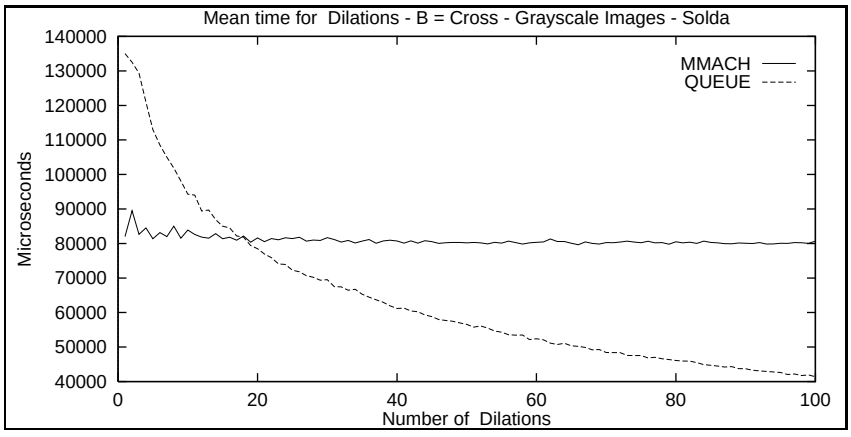

Figura 7.24: Tempo médio da dilatação para a imagem solda, B = losango

7.6.2 Resultados para a Dilatação Condicional

Nestes experimentos usamos como imagem marcadora para a n-dilatação condicional o resultado de uma n-erosão da imagem máscara. Além disso, a dilatação condicional é feita usando como elemento estruturante o mesmo elemento que usamos na n-erosão.

Os algoritmos que analisamos foram:

- MMACH - é o algoritmo de dilatação condicional implementado na MMACH para imagens binárias ou em níveis de cinza em formato byte ou short.
- QUEUE - é um algoritmo de dilatação condicional modular, i.e., que executa a dilatação da imagem e depois um ínfimo com a imagem máscara. Ambos usam a estrutura de fila, apresentado na seção anterior.
- BETTER QUEUE - é um algoritmo de dilatação condicional conjugado, i.e., que executa a dilatação e o ínfimo com a imagem máscara para cada ponto da imagem usando a estrutura de fila apresentada na seção anterior.

Dilatação Condicional em Imagens Binárias

Em cada experimento fizemos de 1 a 17 erosões na imagem máscara (no experimento 2 foi possível fazer de 1 a 20 erosões antes que as componentes conexas desaparecessem) e em seguida o mesmo número de dilatações condicionais na imagem marcadora. Medimos os tempos totais (i.e., o tempo que o processador demorou para calcular n-dilatações condicionais e os tempos médios para cada uma das n-dilatações condicionais.

Experimento 1

Este experimento foi realizado usando como imagem máscara uma imagem binária (fig. 7.5) e como elemento estruturante a cruz.

Os gráficos da fig. 7.25 e da fig. 7.26 mostram, respectivamente, os resultados obtidos para o tempo total de cada n-dilatação condicional e para o tempo médio gasto em cada uma das dilatações condicionais em uma n-dilatação condicional para o experimento 1.

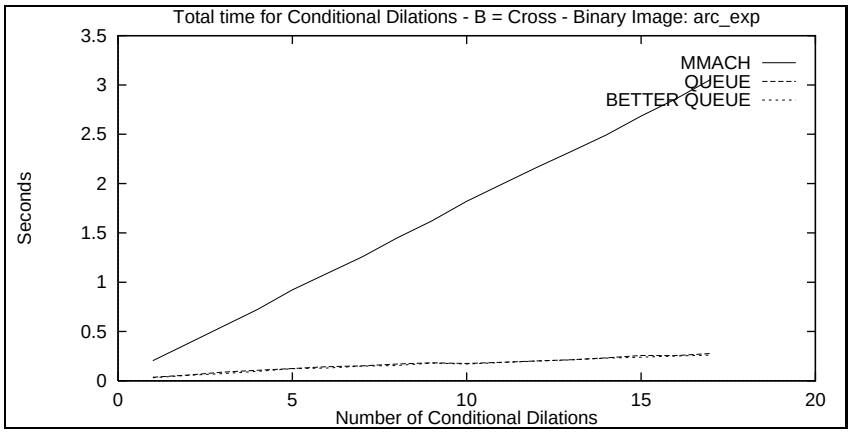

Figura 7.25: Tempo total da dilatação condicional para a imagem arc_exp, $B = \text{cruz}$

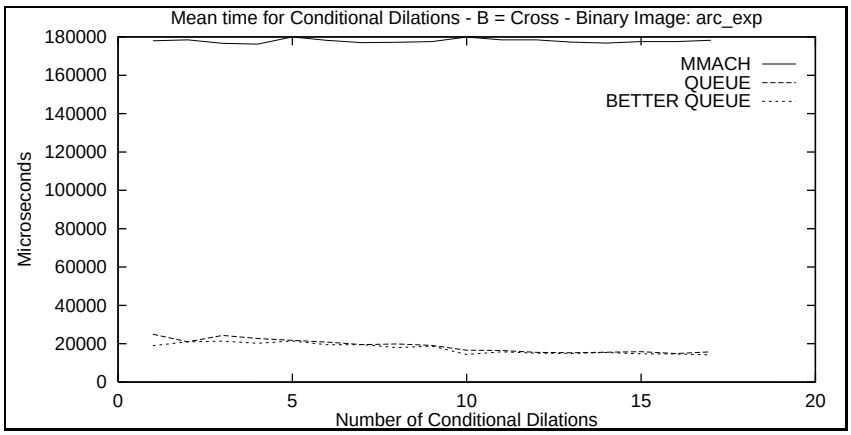

Figura 7.26: Tempo médio da dilatação condicional para a imagem arc_exp, $B = \text{cruz}$

Experimento 2

Este experimento foi realizado usando como imagem máscara uma imagem binária (fig. 7.6) e como elemento estruturante a cruz.

Os gráficos da fig. 7.27 e da fig. 7.28 mostram, respectivamente, os resultados obtidos para o tempo total de cada n-dilatação condicional e para o tempo médio gasto em cada uma das dilatações condicionais em uma n-dilatação condicional para o experimento 2.

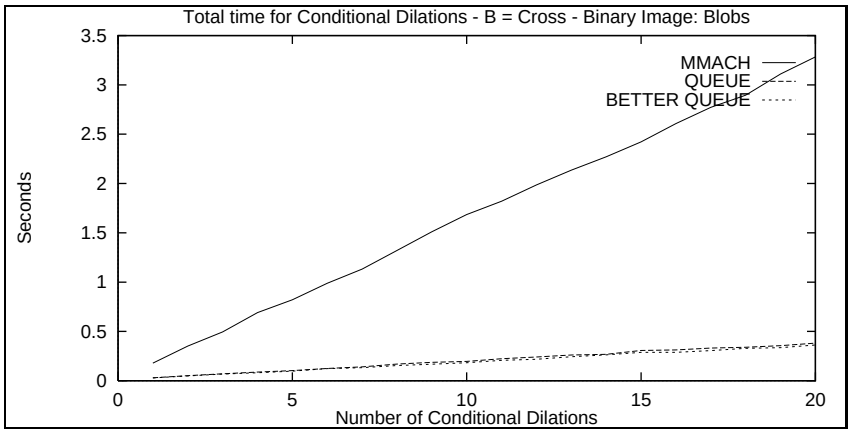

Figura 7.27: Tempo total da dilatação condicional para a imagem blobsgd, B = cruz

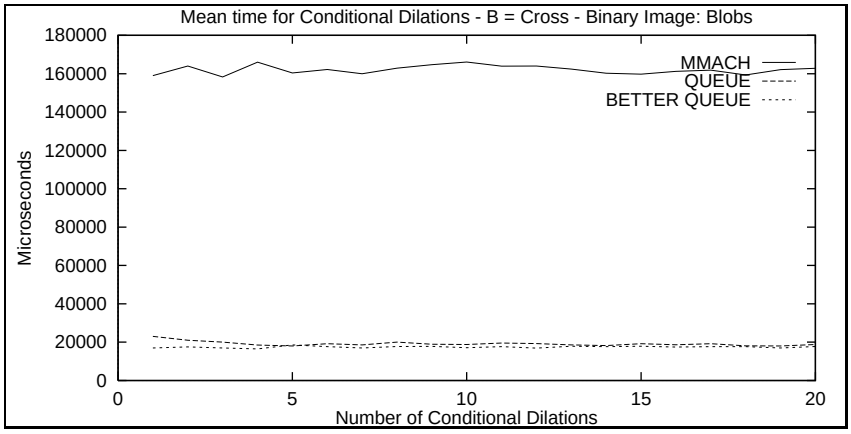

Figura 7.28: Tempo médio da dilatação condicional para a imagem blobsgd, B = cruz

Os resultados dos experimentos 1 e 2 são compatíveis novamente com o esperado. A n-dilatação condicional BETTER-QUEUE é mais rápida que a n-dilatação condicional QUEUE e esta, por sua vez, mais rápida do que a n-dilatação condicional MMACH.

Dilatação Condicional para Imagens em Níveis de Cinza

Fizemos experimentos utilizando como imagens máscara as mesmas imagens que utilizamos para a n-dilatação. Em cada experimento fizemos de 1 a 50 erosões na imagem máscara e em seguida aplicamos a n-dilatação condicional.

Experimento 3

Os gráficos da fig. 7.29 e da fig. 7.30 mostram, respectivamente, os resultados obtidos para o tempo total de cada n-dilatação condicional e para o tempo médio gasto em cada uma das dilatações condicionais em uma n-dilatação condicional para o experimento 3, onde foi usada a imagem da fig. 7.7 e o elemento estruturante cruz.

Neste experimento, o algoritmo BETTER-QUEUE fica mais rápido do que o algoritmo MMACH a partir da vigésima-sexta iteração, enquanto que o algoritmo QUEUE fica melhor a partir da trigésima-primeira iteração.

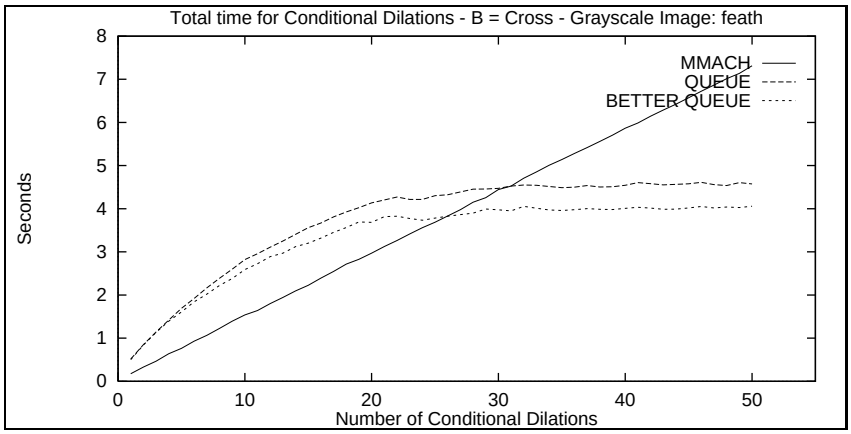

Figura 7.29: Tempo total da dilatação condicional para a imagem feath, B = cruz

Experimento 4

Os gráficos da fig. 7.31 e da fig. 7.32 mostram, respectivamente, os resultados obtidos para o tempo total de cada n-dilatação condicional e para o tempo médio gasto em cada uma das dilatações condicionais em uma n-dilatação condicional para o experimento 4, onde foi usada a mesma imagem que no experimento anterior mas foi mudado o elemento estruturante, que agora é o quadrado elementar.

Neste experimento, o algoritmo BETTER-QUEUE fica mais rápido do que o algoritmo MMACH a partir da vigésima-segunda iteração, enquanto que o algoritmo QUEUE fica melhor a partir da vigésima-nona iteração.

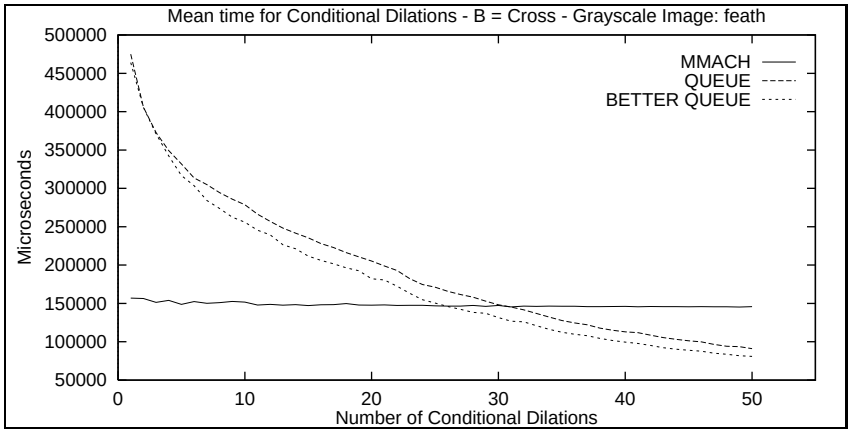

Figura 7.30: Tempo médio da dilatação condicional para a imagem feath, B = cruz

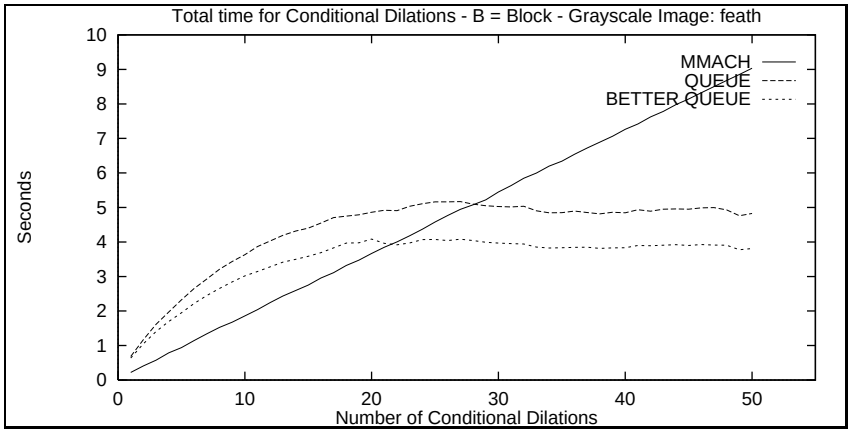

Figura 7.31: Tempo total da dilatação condicional para a imagem feath, B = quadrado elementar

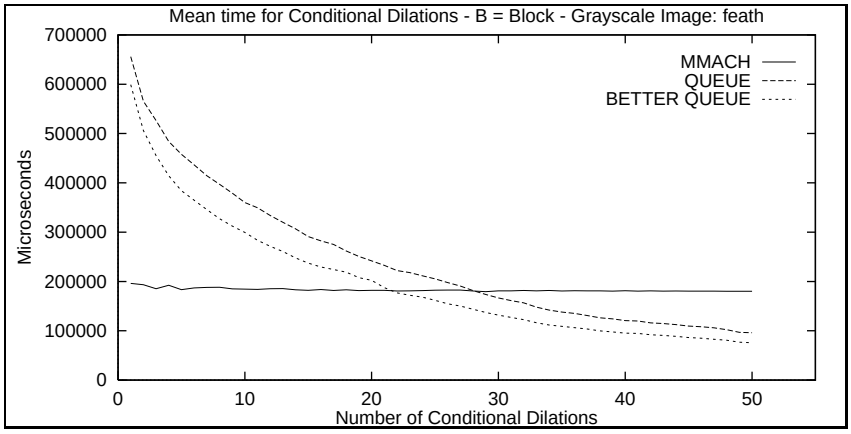

Figura 7.32: Tempo médio da dilatação condicional para a imagem feath, B = quadrado elementar

Experimento 5

Os gráficos da fig. 7.33 e da fig. 7.34 mostram, respectivamente, os resultados obtidos para o tempo total de cada n-dilatação condicional e para o tempo médio gasto em cada uma das dilatações condicionais em uma n-dilatação condicional para o experimento 5, onde usamos a imagem da fig. 7.8 e o elemento estruturante cruz.

Neste experimento, o algoritmo BETTER-QUEUE e o algoritmo QUEUE ficam mais rápidos do que o algoritmo MMACH a partir da terceira iteração. Isso era de se esperar pois a imagem usada (fig. 7.8) tem mais platôs do que a imagem dos dois experimentos anteriores.

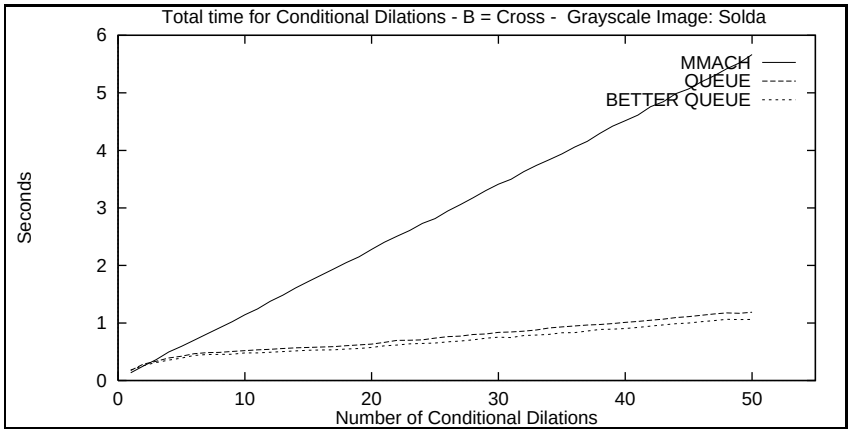

Figura 7.33: Tempo total da dilatação condicional para a imagem solda, B = cruz

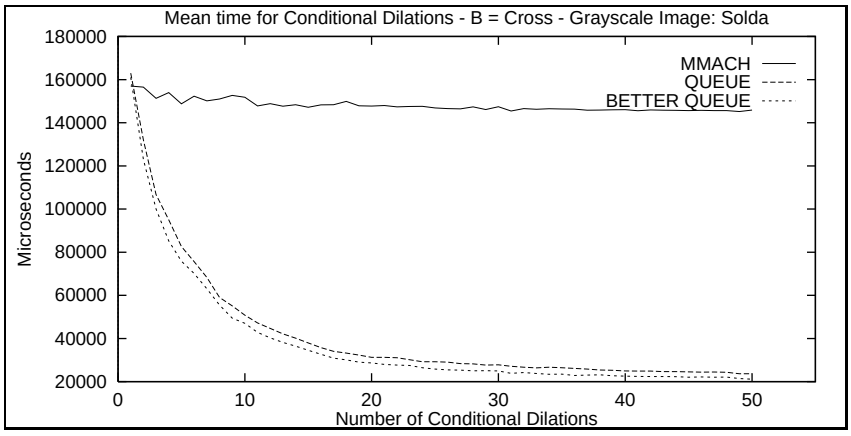

Figura 7.34: Tempo médio da dilatação condicional para a imagem solda, B = cruz

Os gráficos dos tempos totais para a n-dilatação condicional mostram como os algoritmos QUEUE e BETTER-QUEUE ficam mais rápidos do que o algoritmo MMACH mais depressa do que no caso da n-dilatação.

Os gráficos dos tempos médios mostram a queda acentuada do tempo necessário para fazer uma dilatação condicional

Com isso confirmamos novamente a utilidade dos algoritmos que fazem uso de estruturas de filas. Isso porque a dilatação condicional executada pelo algoritmo MMACH tem que varrer a imagem duas vezes, uma vez para a dilatação e outra vez para o ínfimo, enquanto que o algoritmo QUEUE varre duas vezes os pontos que pertencem às bordas da imagem e o algoritmo BETTER-QUEUE varre apenas uma vez esses pontos.

7.6.3 Resultados para a Reconstrução

Estes experimentos são uma repetição dos experimentos da dilatação condicional, mas foram necessários para comparar os nossos algoritmos com o algoritmo de reconstrução rápida que, afinal, motivou os nossos.

Neles também usamos como imagem marcadora para a reconstrução o resultado de uma n-erosão da imagem máscara e como elemento estruturante o mesmo elemento que usamos na n-erosão.

Os algoritmos que analisamos foram:

- MMACH-CLAS - é o algoritmo de reconstrução implementado de acordo com a definição, i.e., iterando a dilatação condicional implementada na MMACH até a estabilidade. No caso das imagens binárias fizemos os experimentos apenas com imagens em formato "byte".
- MMACH - é o algoritmo de reconstrução rápida apresentado no capítulo 6 e que está implementado na MMACH.
- QUEUE - é um algoritmo de reconstrução híbrida implementado de acordo com a definição 7.4.1. Para a segunda parte do algoritmo foram usados os algoritmos de dilatação e ínfimo que usam a estrutura de fila
- BQUEUE - é também um algoritmo de reconstrução híbrida implementado de acordo com a definição 7.4.1. A diferença com o algoritmo QUEUE é que na segunda parte este usa a dilatação condicional conjugada mencionada anteriormente.

Reconstrução em Imagens Binárias

Em cada experimento fizemos de 1 a 17 erosões na imagem máscara (no experimento 2 foi possível fazer de 1 a 20 erosões antes que as componentes conexas desaparecessem) e em seguida aplicamos a reconstrução usando o mesmo elemento estruturante que havíamos usado na erosão. Nestes experimentos medimos apenas os tempos totais (i.e., o tempo que o processador demorou para calcular a reconstrução).

Experimento 1

O gráfico da fig. 7.35 mostra o resultado dos tempos totais dos algoritmos de reconstrução apresentados acima. A imagem utilizada foi a da fig. 7.5 e o elemento estruturante foi a cruz.

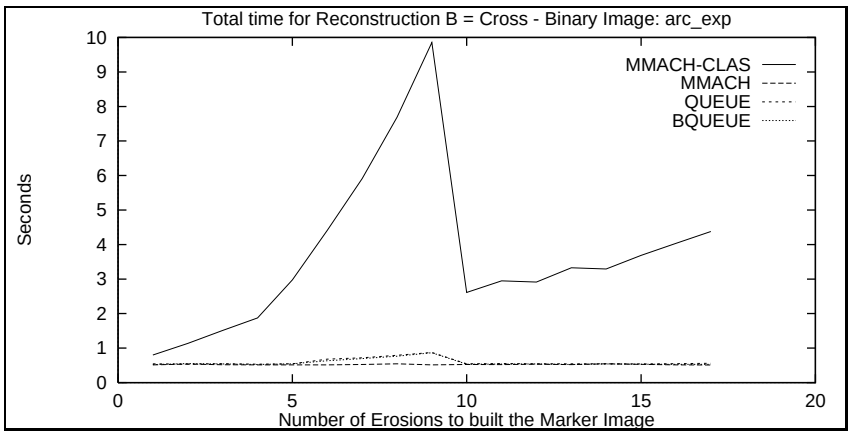

Figura 7.35: Tempo total da reconstrução para a imagem arc_exp, B = cruz

O salto que acontece entre a nona e a décima iteração deste gráfico é devido ao desaparecimento das componentes elípticas da imagem após 10 erosões.

Experimento 2

O gráfico da fig. 7.36 mostra o resultado dos tempos totais dos mesmos algoritmos de reconstrução para a imagem da fig. 7.6, usando o elemento estruturante cruz.

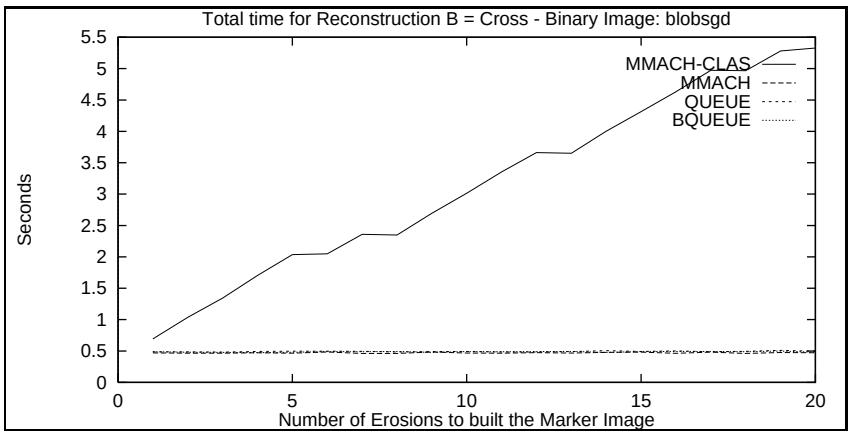

Figura 7.36: Tempo total da reconstrução para a imagem blobsgd, B = cruz

Nestes experimentos com imagens binárias podemos ver que os nossos algoritmos têm performance muito semelhante ao algoritmo rápido de reconstrução. Isto é devido à primeira parte do algoritmo híbrido que é semelhante para todos, com exceção do algoritmo MMACH-CLAS. Como foi mencionado no capítulo 6, são necessárias, em média, poucas varreduras para reconstruir uma imagem binária. No caso da imagem da fig. 7.6 é necessária apenas uma varredura completa, o

que explica o comportamento dos algoritmos apresentados no gráfico da fig. 7.36.

Como era de se esperar, também, o algoritmo de reconstrução implementado da forma clássica tem um desempenho bem pior do que os outros.

Reconstrução em Imagens em Níveis de Cinza

Em cada experimento fizemos de 1 a 50 erosões na imagem máscara e em seguida aplicamos a reconstrução. Nestes experimentos medimos apenas os tempos totais (i.e., o tempo que o processador demorou para calcular a reconstrução).

Experimento 3

O gráfico da fig. 7.37 mostra o resultado dos tempos totais dos algoritmos de reconstrução apresentados acima. A imagem utilizada foi a da fig. 7.7 e o elemento estruturante foi a cruz.

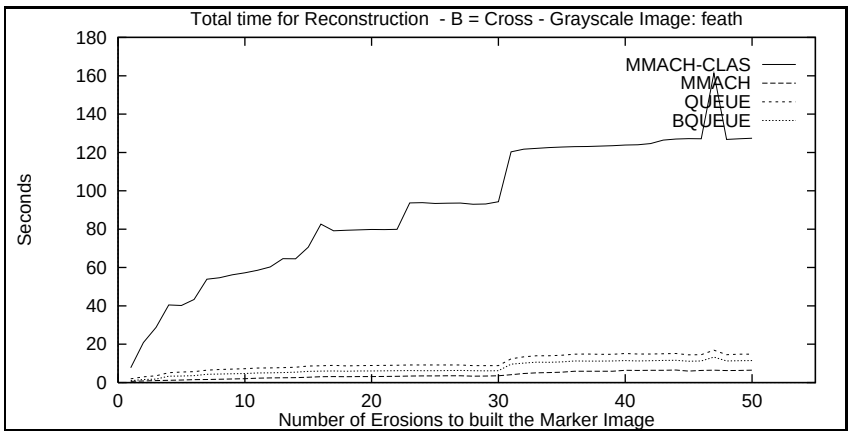

Figura 7.37: Tempo total da reconstrução para a imagem feath, B = cruz

O tempo do algoritmo MMACH é novamente o melhor e o do algoritmo MMACH-CLAS o pior, como era esperado. O tempo dos nossos algoritmos, embora grandes, são bem melhores do que os do algoritmo MMACH-CLAS. Esse tempo pode ser melhorado se, ao invés de uma varredura completa na primeira parte do algoritmo fizéssemos mais. Isso é justificável pois a escolha de uma varredura no algoritmo rápido é experimental e nada impede que seja feita uma outra escolha. Na gráfico da fig. 7.38 mostramos um experimento que fizemos mudando o número de varreduras completas da primeira parte do algoritmo QUEUE antes de iniciar a segunda parte do algoritmo. O algoritmo rotulado "BQUEUE2" realizou 2 varreduras completas e o "BQUEUE5", 5 varreduras completas.

Analisando esse resultado vemos que é possível encontrar uma relação entre o número de pontos de ∂f e o número de varreduras feitas na primeira parte do algoritmo de modo a torná-lo o mais eficiente possível.

Experimento 4

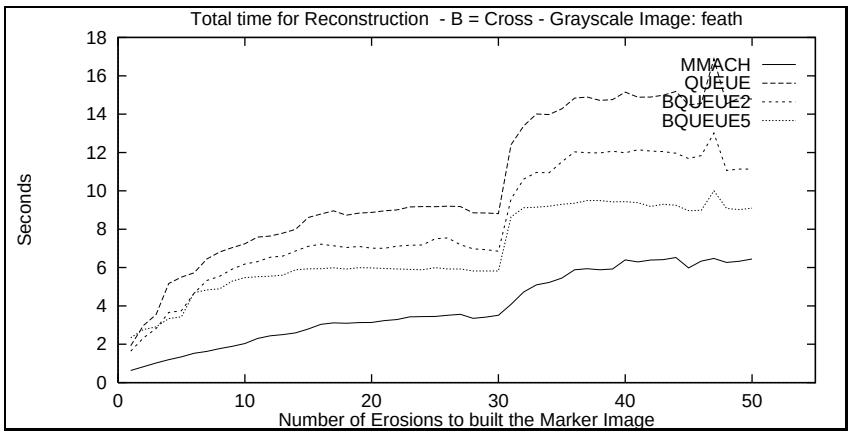

Figura 7.38: Tempo total da reconstrução para a imagem feath, B = cruz - variação de um mesmo experimento

O gráfico da fig. 7.39 mostra o resultado dos tempos totais dos mesmos algoritmos de reconstrução para a mesma imagem do experimento anterior, porém usando o quadrado elementar como elemento estruturante.

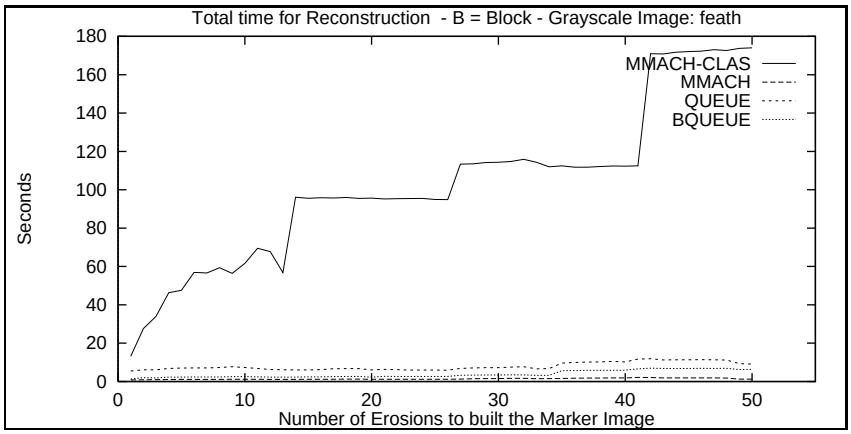

Figura 7.39: Tempo total da reconstrução para a imagem feath, B = quadrado elementar

Experimento 5

O gráfico da fig. 7.40 mostra o resultado dos tempos totais dos algoritmos de reconstrução para a imagem da fig. 7.8, usando o elemento estruturante cruz

Da mesma forma que o resultado da n-dilatação condicional para esta imagem foi melhor do que para a imagem da fig. 7.7, aqui também isso se confirma.

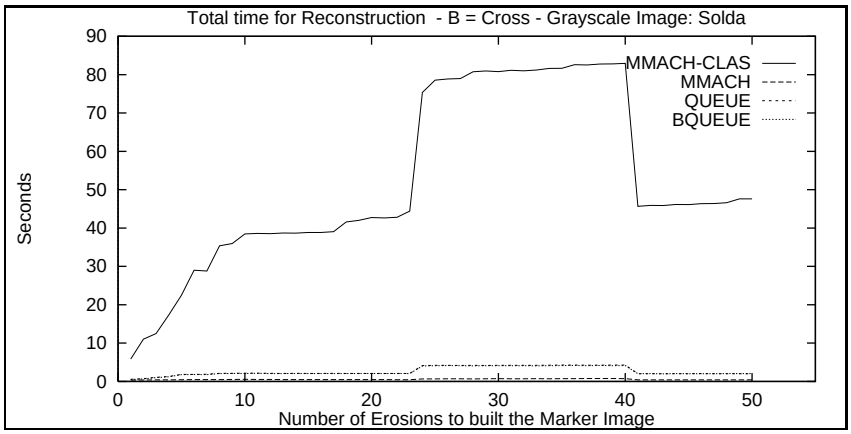

Figura 7.40: Tempo total da reconstrução para a imagem solda, B = cruz

Capítulo 8

Conclusão

The end crowns all,
And that old arbitrator, Time,
Will one day end it.

W. Shakespeare

Dentro da área de processamento de imagens, relacionada à segmentação, vimos que a abordagem clássica, não estatística, fornece-nos muitas técnicas para a solução de problemas específicos, porém, não é uma abordagem geral, o que a torna desinteressante sob muitos aspectos, inclusive em termos de implementação do "software" para realizar as transformações entre imagens.

A abordagem clássico-estatística é muito poderosa e tem sido usada com muito sucesso em diversas aplicações práticas importantes como OCR, reconhecimento de sinais, reconhecimento de faces e etc. Porém, ela também não é geral e por isso também não é uma solução definitiva para qualquer problema de segmentação.

A Morfologia Matemática é uma abordagem que tem ganho cada vez mais adeptos no mundo todo, principalmente por ser uma teoria algébrica baseada em poucas operações e operadores elementares capazes expressar quaisquer operadores entre imagens, os quais podem ser facilmente implementados.

Na primeira parte deste trabalho procuramos mostrar as potencialidades desta teoria aplicada a segmentação de imagens. Apresentamos diversos exemplos de como usar os operadores morfológicos em problemas reais de segmentação de imagens, além de mostrarmos a equivalência entre a abordagem clássica e a abordagem morfológica.

Cremos que nossa maior contribuição nessa primeira parte é a criação de um texto sobre segmentação de imagens, sob o ponto de vista dessa teoria, contendo uma coletanea bastante ampla de exemplos, e que pode ser usado como parte de um curso de graduação ou pós-graduação

em processamento de imagens. Além disso, colocamos boa parte desse texto em formato html e disponibilizamos para o acesso via "Web".

Embora possamos implementar facilmente as operações e operadores elementares da *MM* e construir a partir deles quaisquer operadores entre imagens, vimos que essa abordagem pode comprometer a performance dos operadores que são formados por uma composição complexa das operações e operadores elementares.

Na segunda parte do trabalho mostramos algoritmos rápidos que imitam o funcionamento de operadores importantes de segmentação de imagens. Nessa parte nossa contribuição foi a implementação desses algoritmos na toolbox MMACH. Todos os exemplos mostrados no capítulo de aplicações foram solucionados utilizando-se esses algoritmos rápidos.

Como estes algoritmos rápidos mudam a arquitetura do "software" que implementa a MMACH (capítulo 7), vimos a necessidade de desenvolvermos algoritmos rápidos para os operadores e operações elementares da *MM*. Para isso usamos as idéias básicas dos algoritmos rápidos que implementam operadores complexos como a reconstrução.

Outra contribuição deste trabalho foi mostrar que é possível, usando implementações eficientes dos operadores elementares, implementar operadores complexos da *MM* melhorando bastante a eficiência do operador. Isso foi feito usando-se a estrutura de dados do tipo fila (FIFO) na implementação os operadores elementares. Embora a eficiência em relação aos algoritmos rápidos seja um pouco inferior, há duas vantagens importantes dessa abordagem :

1. Não temos que mudar a arquitetura do "software" que implementa a máquina mofológica (mantêm-se a modularidade).
2. Esses operadores podem ser usados para implementar outros operadores complexos. Assim, aproveita-se a flexibilidade da arquitetura da MMACH para ganhar eficiência em outros operadores morfológicos.

Como proposta de pesquisas futuras, vemos algumas linhas de trabalho que ainda podem ser exploradas. Na área de segmentação, embora seja possível (em teoria) construir um operador morfológico para segmentar qualquer imagem, não existe uma metodologia bem definida para resolver qualquer problema de segmentação. Este é um problema em aberto cuja solução é difícil no estado da arte atual.

Quanto à segmentação de imagens coloridas, ainda não está bem claro como modelar uma imagem colorida em termos de reticulados completos. A segmentação de imagens coloridas ainda é feita sobre cada uma das bandas da imagem, combinando-se posteriormente as informações. Por outro lado, é razoável supor que uma imagem colorida seja mais simples de segmentar do que uma imagem em níveis de cinza, por conter mais informações do que a outra. Neste sentido, achamos que esse deve ser um campo onde ainda há muito o que pesquisar.

Outra linha de pesquisa nessa área é o desenvolvimento de uma teoria de segmentação de imagens. Isto também deve ser um assunto muito difícil pois, como havíamos citado anteriormente, depende fortemente do modelamento algébrico e estatístico da imagem.

Não descartamos também a pesquisa de novos algoritmos rápidos que imitem operadores

morfológicos complexos pois este estudo ainda é muito importante para aplicações comerciais e industriais e nesses casos, pode-se pagar o preço do desenvolvimento desse "software".

Apêndice A

O sistema KHOROS

O KHOROS é um sistema de desenvolvimento, manutenção e execução de programas, utilizado para o processamento de imagens em ambientes UNIX [RASW90, KR94, RJL94]. Possui, dentre outras coisas, ferramentas de especificação de "interfaces" com o usuário, de desenvolvimento e manutenção de programas e uma "interface" de programação visual com acesso a bibliotecas prontas; além da possibilidade de acrescentar-se novos pacotes de programas para processamento de imagens. O pacote completo pode ser conseguido via ftp no seguinte endereço: **ftp.khoral.com**; ou via http no endereço: **http://www.khoral.com/.**

O Khoros possui um sistema de auxílio à programação ("**composer**") (veja a imagem da fig. A.1) que contém as ferramentas de especificação de programas, "interfaces", documentação e etc.

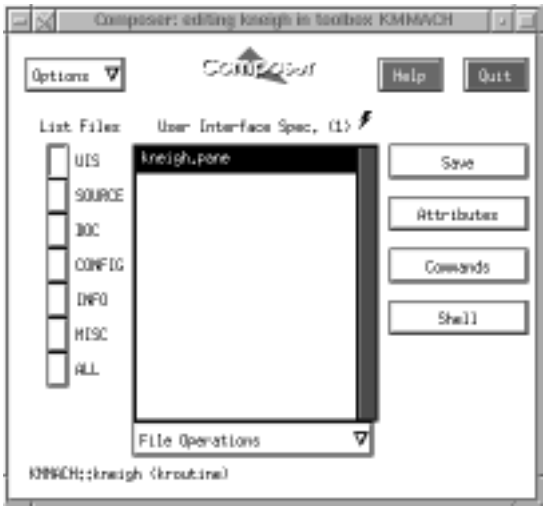

Figura A.1: Ambiente de Auxílio à Programação

Por exemplo, no KHOROS, o projeto de um programa requer primeiramente o planejamento

de uma "interface " com o usuário. A especificação da GUI ("graphic user interface") é feita com uma ferramenta do "composer" chamada "Guise", que gera uma descrição de alto nível das entradas, parâmetros e saídas. Esta descrição é armazenada num arquivo com extensão "**.pane**". Abaixo apresentamos um exemplo de um arquivo de especificação de uma GUI.

```
-F 4.3 1 0 52x1+0+0 +0+0 ´Cantata´ cantata
  -M 1 1 52x1+0+0 +1+0 ´Main driver of dilation, erosion, opening, closing´ subform
    -P 1 0 52x1+0+1 +0+0 ´ ´ kdil_ero
      -D 1 0 9x1+0+0 ´Options´ _gui_options
        -H 1  6x1+0+0 ´License´ ´license´ \$KMMACH/repos/license/License license
      -E
      -R 1 0 1 5x1+35+0 ´Run´ ´execute operation´ \$KMMACHBIN/kneigh
      -H 1  5x1+41+0 ´Help´ ´help page´ \$KMMACH/objects/kroutine/kneigh/help help
      -Q 1 0 5x1+47+0 ´Close´
      -I 1 0 0 1 0 0 52x1+0+1.5 ´ ´ ´Input Image ´ ´Input Image´ i
      -I 1 0 0 1 0 0 52x1+0+3 ´ ´ ´Struct Elem.´ ´Structuring Element´ str
      -O 1 0 0 1 0 0 52x1+0+4.5 ´ ´ ´Output Image´ ´Output Image´ o
      -C 1
        -t 1 0 1 1 0 8.375x1+0+6 ´Dilation´ ´dilation option flag´ dil
        -t 1 0 1 0 0 8.375x1+10+6 ´Anti-dil´ ´anti-dilation option flag´ adil
        -t 1 0 1 0 0 8.375x1+19+6 ´Opening ´ ´opening option flag´ open
        -t 1 0 1 0 0 11.375x1+33+6.5 ´Morph. Grad.´ ´morph. gradient option flag´ grad
        -t 1 0 1 0 0 8.375x1+0+7.5 ´Erosion ´ ´erosion flag option´ ero
        -t 1 0 1 0 0 8.375x1+10+7.5 ´Anti-ero´ ´anti-erosion option flag´ aero
        -t 1 0 1 0 0 8.375x1+19+7.5 ´Closing´ ´closing option flag´ close
        -t 1 0 1 0 0 11.375x1+0+9.5 ´Morph. Skel.´ ´morphological skeleton option flag´ mskel
        -t 1 0 1 0 0 11.375x1+0+11 ´Last Erosion´ ´last erosion option flag´ lastero
        -t 1 0 1 0 0 11.375x1+14+9.5 ´Local Maximum´ ´local maximum option flag´ locmax
        -t 1 0 1 0 0 12.125x1+14+11 ´Local Minimum´ ´local minimum option flag´ locmin
      -E
    -E
  -E
-E
```

Esta especificação é usada para gerar o código para as "interfaces" visuais ou de linha de comando. Nesse código gerado, já estão inclusas as rotinas de validação das entradas. Além disso, ela serve para a apresentação gráfica da interface com o usuário (GUI), assim como um ícone correspondente. A imagem da fig. A.2 mostra a GUI da especificação acima, e a imagem da fig. A.3 mostra o ícone resultante.

A ferramenta mais importante do KHOROS é o "**cantata** ", (veja a imagem da fig. A.4). Cantata é o ambiente visual de programação de alto nível para a solução de um problema de processamento de imagens. Neste ambiente é possível colocar as rotinas de manipulação de imagens ligadas na sequência desejada, construir malhas de repetição (while-loop, count-loop) e estruturas de decisão (if-then-else).

Cada programa é representado por um ícone, que no cantata é chamado "glyph" (veja figura A.3), que possui, do lado esquerdo, terminais de entrada e, do lado direito, terminais de saída, representados por pequenos quadrados. Ao centro, há o botão para a execução do programa e

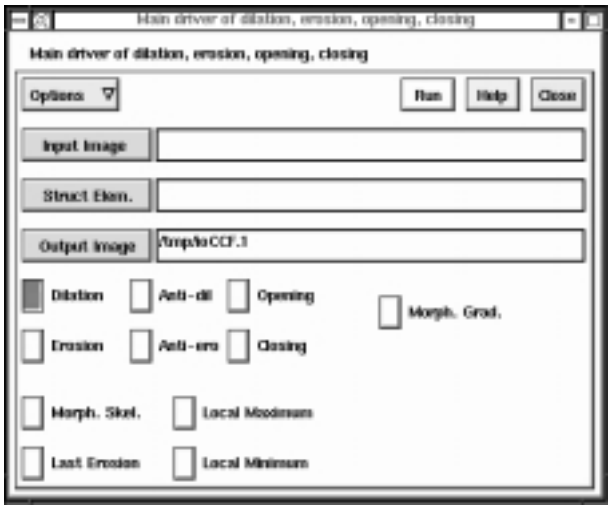

Figura A.2: Interface gráfica gerada pelo arquivo “pane”

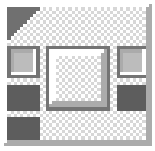

Figura A.3: Representação visual de um arquivo “pane”

no canto superior esquerdo um botão triangular que apresenta o painel gráfico para a edição dos parâmetros (veja a imagem da fig. A.2).

As ligações entre os “glyphs” são representadas por linhas sólidas entre os terminais de saída de um programa com as entradas de outros, como se fossem dutos por onde passarão os dados. Para fazer essas ligações, basta “clicar” o botão do terminal de saída de um programa e o botão de entrada do outro.

Cada vez que um “glyph” é executado, um processo UNIX é disparado como se fosse executado via linha de comando. O processo pode se acompanhado na janela de exibição de comandos, parte inferior da imagem da fig. A.4. Em ambientes ligados em rede, esses “glyphs” podem ser executados em máquinas remotas, simulando um processamento paralelo da aplicação.

Ao programa formado pelo conjunto das rotinas (“glyphs”), nesse ambiente, é dado o nome de “**workspace**”. Esse “workspace” pode ser encapsulado em um “glyph” para ser usado como uma subrotina de uma outra aplicação (por exemplo, se o “workspace” serve para filtrar uma imagem, ele pode encapsulado e usado em um outro “workspace”). Além disso, ele pode ser executado fora do ambiente do cantata como se fosse um “shell-script”.

A versão atual do KHOROS é a 2.1, a versão que desenvolvemos nosso trabalho é a versão 1.0.5. Ambas podem ser portadas para **LINUX** e têm performance semelhante ou até melhor, em alguns casos, que quando usadas num ambiente **SUN** em rede.

A MMACH é uma "toolbox" que contém diversos operadores morfológicos e foi desenvolvida num esforço conjunto da USP, UNICAMP e INPE. O nome MMACH vem de máquina morfológica que, como vimos, é o nome dado a uma implementação da linguagem morfológica. Ela foi desenvolvida para a versão 1.0.5 do KHOROS e já foi parcialmente portada para a versão 2.1 [RJL94]. Ambas versões podem ser conseguidas via ftp nos endereços: ftp.ime.usp.br e ftp.fee.dca.unicamp.br. A documentação de como compilar e instalar a "toolbox" acompanha o pacote. Ainda como parte de nosso trabalho fizemos uma documentação das rotinas mais importantes da MMACH, incluindo exemplos do seu uso, em formato HTML, para ser usado de forma integrada com um "Browser" para a WWW. Esta documentação é distribuída juntamente com a MMACH.

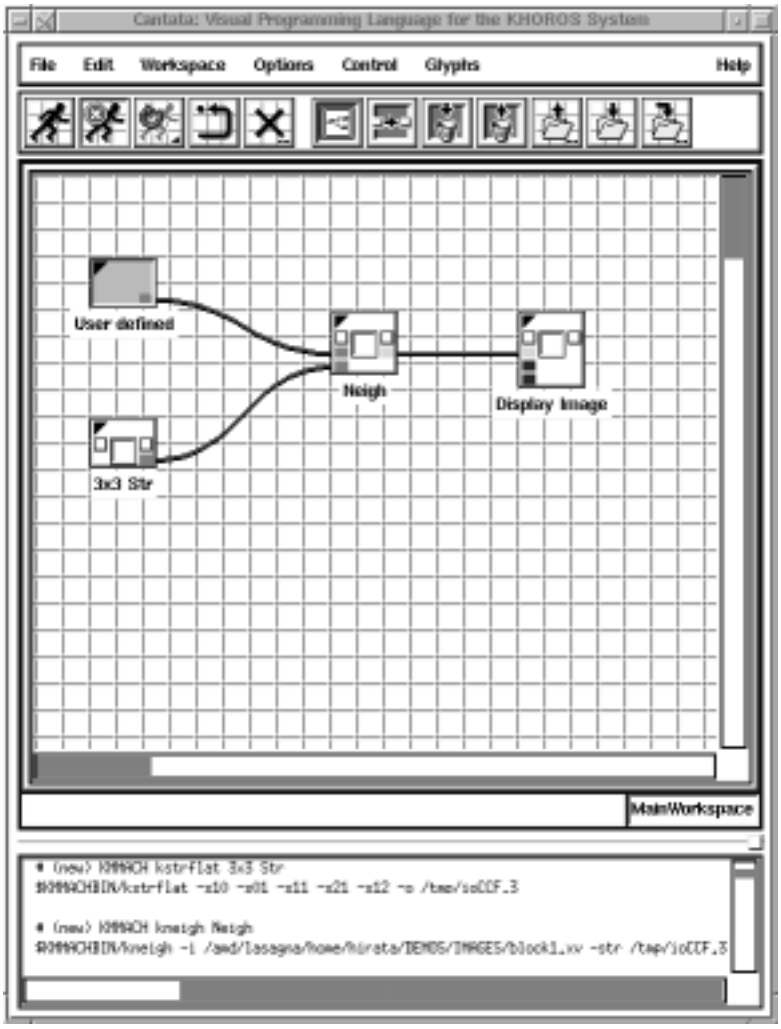

Figura A.4: Ambiente de Programação Visual

Bibliografia

[Apo58] Tom Apostol. *Mathematical Analisys*. Addison-Wesley, 1958.

[Ban95] G. J. F. Banon. Characterization of translation-invariant elementary morphological operators between gray-level images. Technical report, INPE, 1995.

[Bax94] G. A. Baxes. *Digital Image Processing - Principles and Applications*. John Wiley and Sons, Inc., 1994.

[BB91] G. J. F. Banon and J. Barrera. Minimal representation for translation-invariant set mappings by mathematical morphology. *SIAM J. Appl. Math.*, 51:1782–1798, 1991.

[BB92] J. Barrera and G. J. F. Banon. Expressiveness of the morphological language. In *Image Algebra and Morphological Image Processing III*, volume 1769, pages 264–275, San Diego, California, 1992. SPIE.

[BB93] G. J. F. Banon and J. Barrera. Decomposition of mappings between complete lattices by mathematical morphology: Part I. 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, Pernambuco, 1994.

[BBL94] J. Barrera, G. Banon, and R. Lotufo. A Mathematical Morphology Toolbox for the KHOROS System. In *Image Algebra and Morphological Image Processing V*, California, july 1994. SPIE Proceedings.

[BdSB94] J. Barrera, F.S.C. da Silva, and G. J. F. Banon. Automatic Programming of Binary Morphological Machines. In *Image Algebra and Morphological Image Processing*, volume 2300, pages 229–240, San Diego, 1994. SPIE Proceedings.

[Beu90] S. Beucher. *Segmentation d'Images et Morphologie Mathématique*. PhD thesis, Ecole Nationale Supérieure des Mines de Paris, Fontainebleau, 1990.

[BF70] C. R. Brice and C. L. Fenema. Scene analysis using regions. *Artificial Intelligence*, 1:205–226, 1970.

[Bir67] G. Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, Rhode Island, 1967.

[BM93] S. Beucher and F. Meyer. The morphological approach to segmentation: the watershed transformation. In E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, chapter 12, pages 433–481. Marcel Dekker, New York, 1993.

[BTdST95] J. Barrera, N. S. Tomita, F. S. C. da Silva, and R. Terada. Automatic Programming of Binary Morphological Machines by PAC Learning. In *Neural and Stochastic Methods in Image and Signal Processing*, volume 2568, pages 233–244, San Diego, 1995. SPIE.

[BTdST96] J. Barrera, R. Terada, F. S. C. da Silva, and N. S. Tomita. Automatic Programming of Morphological Machines for OCR. In P. Maragos, R. W. Schafer, and M. A. Butt, editors, *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 385–392, Atlanta, GA, May 1996. International Symposium on Mathematical Morphology, Kluwer Academic Publishers.

[CA79] Guy B. Coleman and Harry C. Andrews. Image segmentation by clustering. In *Proceedings of the IEEE*, volume 67, May 1979.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to algorithms. *McGraw-Hill*, 1990.

[Dav90] E.R. Davies. *Machine Vision: Theory, Algorithms and Practicalities.* Academic Press, 1990.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, 1973.

[d'O92] Marcos Cordeiro d'Ornellas. Algoritmos rápidos para o processamento morfológico de imagens binárias. Master's thesis, ITA, 1992.

[DV84] Nelson Delfino D'Avila and Flavio R. D. Velasco. *Processamento Digital de Imagens.* IME-USP, 1984.

[GW92] R. C. Gonzalez and R. E. Woods. *Digital Image Processing.* Addison-Wesley Publishing Company, 1992.

[Hal79] Ernest L. Hall. *Computer Image Processing and Recognition.* Academic Press, 1979.

[Har92] Robert Haralick. *Encyclopedia of Artificial Intelligence*, volume 2, chapter Segmentation, pages 1473–1491. John Wiley and Sons, Inc., 1992.

[HDW94] J.Flum H.-D.Ebbinghaus and W.Thomas. *Mathematical Logic.* Springer Verlag, 1994.

[Hei91] H. J. A. M. Heijmans. Theoretical aspects of gray-level morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:568–582, 1991.

[Hei95] H. J. A. M. Heijmans. Composing morphological filters. Research report BS-R9504, CWI, 1995. Submitted to IEEE Trans. Image Proc.

[HS85] Robert M. Haralick and Linda G. Shapiro. Image segmentation techniques. *Computer Vision, Graphics and Image Processing*, 29:100–132, 1985.

[IS56] E. J. Isaac and R. C. Singleton. Sorting by Address Calculation. *Journal ACM*, 3:169–174, 1956.

[Kan80] Takeo Kanade. Region segmentation: Signal vs. semantics. *Computer Vision and Image Processing*, (13):279–297, 1980.

[KR89] T. Y. Kong and A. Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics and Image Processing*, (48):357–393, 1989.

[KR94] K. Konstantinides and J. Rasure. The KHOROS Software Development Environment for Image and Signal Processing. *IEEE Transactions on Image Processing*, 3(3):243–252, 1994.

[LM84] C. Lantuéjoul and F. Maisonneuve. Geodesic methods in quantitative image analysis. *Pattern Recognition*, 17:177–187, 1984.

[Mar82] David Marr. *Vision*. W.H.Freeman, 1982.

[MB90] F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication and Image Representation*, 1:21–46, 1990.

[Mey91] F. Meyer. Skeletons and perceptual graphs. *Signal Processing*, 16:335–363, 1991.

[Min03] H. Minkowski. Volumen und Oberfläche. *Math. Ann.*, 57:447–495, 1903.

[Pra91] W. K. Pratt. *Digital Image Processing*. John Wiley and Sons, 1991.

[RASW90] J. Rasure, D. Argiro, T. Sauer, and C. Williams. Visual Language and Software Development Environment for Image Processing. *International Journal of Imaging Systems and Technology*, 2:183–199, 1990.

[RD79] Azriel Rosenfeld and Larry S. Davis. Image segmentation and image model. In *Proceedings of the IEEE*, volume 67, May 1979.

[RJL94] J. Rasure, R. Jordan, and R. A. Lotufo. Teaching Image Processing with KHOROS. In *IEEE International Conference on Image Processing*, pages 13–16, 1994. available as postscript at ftp://www.khoral.com/pub/khoros/papers/kieee94.ps.gz.

[Ros69] A. Rosenfeld. *Picture Processing by Computer*. Academic Press, 1969.

[Ros78] A. Rosenfeld. Geodesics in Digital Pictures. *Information and Control*, 36(1):74–84, 1978.

[RP66] A. Rosenfeld and J. L. Pfaltz. Sequential Operations in Picture Processing. *Journal of the Association for Computing Machinery*, pages 471–494, 1966.

[Rus91] John C. Russ. *Computer-Assisted Microscopy*. Plenum Press, New York, 1991.

[Ser82] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[Ser88] J. Serra, editor. *Image Analysis and Mathematical Morphology. II: Theoretical Advances.* Academic Press, London, 1988.

[Ser94] J. Serra. Morphological filtering: an overview. *Signal Processing*, 38:3–11, 1994.

[SS95] P. Salembier and J. Serra. Flat zones filtering, connected operators, and filters by reconstruction. *IEEE Transactions on Image Processing*, 4(8):1153–1160, 1995.

[SV92] J. Serra and L. Vincent. An overview of morphological filtering. *Circuits, Systems and Signal Processing*, 11:47–108, 1992.

[TG74] Julius T. Tou and Rafael C. Gonzalez. *Pattern Recognition Principles.* Addison-Wesley Publishing Company, 1974.

[VB89] L. Vincent and S. Beucher. The morphological approach to segmentation: an introduction. Technical report, Ecole Nationale Supérieure des Mines de Paris, 1989.

[Vin90] L. Vincent. *Algorithmes Morphologiques a Base de Files d'Attente et de Lacets. Extension aux Graphes.* PhD thesis, Ecole Nationale Supérieure des Mines de Paris, Fontainebleau, 1990.

[Vin91] L. Vincent. Morphological transformations of binary images with arbitrary structuring elements. *Signal Processing*, 22:3–23, 1991.

[Vin93a] L. Vincent. Morphological algorithms. In E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, chapter 8, pages 255–288. Marcel Dekker, New York, 1993.

[Vin93b] L. Vincent. Morphological grayscale reconstruction in image analysis: Applications and efficient algorithms. *IEEE Trans. on Image Processing*, 2(2):176–201, April 1993.

[VS91] L. Vincent and P. Soille. Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, June 1991.

[VV88] L. J. Vliet and J. H. Verwer. A contour processing method for fast binary neighbourhood operations. *Pattern Recognition Letters*, 7(1):27–36, January 1988.

[Wat72] S. Watanabe. *Frontiers of Pattern Recognition*, chapter Boundary Detection of Radiographic Images by a Thresholding Method. Academic Press, Inc, 1972.

[YB89] S.D. Yankowitz and A.M. Bruckstein. A new method for image segmentation. *Computer Vision and Image Processing*, 46(1):82–95, April 1989.

[YC74] Tzay Y. Young and Thomas W. Calvert. *Classification, Estimation and Pattern Recognition.* American Elsevier Pub. Co., Inc., 1974.