Erosões e Dilatações Morfológicas Binárias Seqüenciais Rápidas

Anderson Fraiha Machado

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Ronaldo Fumio Hashimoto

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro do CNPq

São Paulo, fevereiro de 2008

Resumo

A Morfologia Matemática (MM) é um arcabouço geral para o estudo de mapeamentos entre imagens binárias. Estes estudos são de especial interesse na área de Processamento de Imagens. Tais mapeamentos entre imagens binárias são conhecidos como operadores de conjunto. Um aspecto importante da MM é a representação destes operadores em termos de dilatações, erosões e outras operações usuais de conjunto (interseção, união, complemento e diferença). Por este motivo, a dilatação e a erosão são ditos operadores morfológicos elementares.

Este trabalho visa propor novos métodos para calcular a erosão e a dilatação morfológica binária rapidamente. Tais métodos se fundamentam em conceitos e técnicas de pré-processamento (em tempo linear) introduzidas por este trabalho, como a transformada da densidade, ou ainda, um conjunto de cascas. O resultado destes pré-processamentos é traduzido em ganho de velocidade dos algoritmos de erosão e dilatação, além de apresentar uma representação compacta dos conjuntos operandos.

O consumo de tempo dos métodos propostos é no pior caso quadrático, porém, num estudo experimental preliminar, o algoritmo se comporta eficientemente, chegando a ser até mesmo linear em alguns casos. Além disso, um levantamento sucinto de outros métodos de erosão e dilatação morfológica binária conhecidos pela literatura atual é apresentado. Algumas simulações e uma breve análise de complexidade mostram que nossos métodos são uma boa alternativa para implementação de erosão e dilatação morfológica eficiente.

Palavras-chave: erosão rápida, dilatação rápida, morfologia matemática, imagem binária.

Sumário

Capítulo 1

Introdução

Atualmente, o *processamento de imagens digitais* pelos computadores é uma atividade eminente e indispensável na vida cotidiana. Ele está presente desde a confecção de convites de casamento até o complexo processamento de imagens de satélite.

Em particular, o processamento de *imagens binárias* é de grande interesse em diversos ramos científicos e industriais. Por exemplo, uma certa indústria precisa controlar a qualidade de seus produtos. Um bom indicativo para isto pode ser obtido avaliando a forma e tamanho da silhueta de seus produtos. Um sistema de controle de qualidade pode ser fundamental para tal indústria. A avaliação automática de produtos requer computadores equipados de softwares de processamento de imagens binárias e ferramentas de aquisição de imagens (como câmeras digitais e esteiras automatizadas). Um outro exemplo pode ser encontrado no ramo da microbiologia: Um grupo de pesquisadores estuda a ploriferação de uma certa colônia de bactérias. Uma imagem binária extraída pelo microscópio computadorizado deve ser analisada precisamente. O reconhecimento de formas por meio de filtros "janelados" é uma boa alternativa para tal análise. Enfim, muitas tarefas em diversas áreas, como processamento de imagens médicas, microbiologia computacional, reconhecimento de formas em geral, análise de texturas, reconhecimento óptico, restauração de imagens e outras têm como objetivo a rapidez e a precisão na análise das imagens digitais, e em particular, imagens binárias. Muitas destas tarefas tipicamente são realizadas em tempo real.

A *Morfologia Matemática* (MM), fundada por J. Serra e G. Matheron ([35, 36]) entre os anos de 1960 e 1969, ingressa neste contexto, tornando-se uma disciplina muito importante na área de Processamento de Imagens. Seu domínio de atuação cobre um largo espectro de aplicações em Processamento de Imagens que vai desde o tratamento de imagens microscópicas até o processamento de imagens de satélites, passando por aplicações industriais, médicas, automação de escritório, etc.

Basicamente, a MM é uma teoria que estuda transformações de formas presentes nas imagens. Estas transformações podem ser vistas como mapeamentos entre reticulados completos [20, 36].

Em particular, mapeamentos entre imagens binárias são de especial interesse em MM. Estes mapeamentos podem ser vistos como sendo mapeamentos entre conjuntos [9, 20, 35] e por esta razão são chamados de *operadores de conjunto*.

O paradigma central da MM é decompor uma classe de operadores morfológicos (W-operadores) em termos de dilatação, erosão e operações usuais de conjunto (união, interseção e complemento).

Sendo assim, a *dilatação* e a *erosão* são tidos como pilares para os outros operadores morfológicos binários. Embora não se conheça métodos para dilatação e a erosão que consomem tempo computacional não-linear, existem alguns métodos que executam tais operações em tempo satisfatório.

Uma importante contribuição deste trabalho consiste em apresentar métodos que executam rapidamente a *erosão morfológica binária* e a *dilatação morfológica binária*.

O Capítulo 2 apresenta uma breve revisão dos conceitos de imagens binárias aplicados à MM, bem como fundamentos matemáticos e morfológicos essenciais para a apresentação de novos métodos para calcular dilatação e erosão. Além disso, neste mesmo capítulo exploraremos um pouco mais esses dois operadores e suas relações com outros bem conhecidos na MM. Tais conceitos apresentados fundamentam as técnicas de pré-processamento em tempo linear introduzidas no Capítulo 3. Tais técnicas são fundamentais para a apresentação dos novos métodos de erosão e dilatação. Serão apresentadas basicamente duas técnicas de pré-processamento nas imagens de entrada: transformação da densidade e um extrator de contorno. Os novos métodos para calcular a dilatação e a erosão rápida (bem como uma breve revisão de outros métodos bem conhecidos que implementam estes) são exibidos no Capítulo 4. Finalmente, no Capítulo 5 mostraremos um curto e objetivo experimento que envolve os mais rápidos métodos seqüenciais que implementam a dilatação e a erosão.

Capítulo 2

Considerações Preliminares

Neste capítulo apresentaremos uma breve revisão de alguns conceitos essenciais da Morfologia Matemática Binária e Processamento de Imagens Binárias, a fim de fundamentar os métodos de dilatação e erosão propostos por este trabalho.

A Seção 2.1 define os conceitos gerais de processamento de imagens binárias; na Seção 2.2 encontra-se uma breve revisão de conceitos primordiais da MM; a Seção 2.3 define algumas relações entre operadores de conjunto, trazendo definições sucintas e propriedades da dilatação e da erosão morfológica binária; e finalmente, a Seção 2.4 exemplifica alguns operadores morfológicos bem conhecidos da MM com suas respectivas decomposições em termos de dilatação e erosão.

2.1 Imagens Binárias

Uma imagem binária em $\mathbb{Z}^n$ pode ser definida como uma função $f : \mathbb{Z}^n \to \{0, 1\}$. Ou seja, uma imagem binária contém apenas dois valores: o *background* (ou *fundo*) da imagem ou o *foreground* (ou *objeto*) da imagem binária.

Seja $x \in \mathbb{Z}^n$ um ponto qualquer da imagem binária f. Se $n = 2$ então dizemos que x é um *pixel* de f, caso contrário, se $n \geq 3$ então dizemos que x é um *voxel* de f. Considere que se x pertence ao objeto da imagem binária f, então $f(x) = 1$. De igual modo, se x pertence ao fundo da imagem binária f, então $f(x) = 0$.

O domínio de uma imagem binária f, denotado por D_f, é definido como o menor hipercubo que envolve f. No espaço $\mathbb{Z}^2$, convenciona-se que D_f é o menor retângulo que envolve a imagem f (o *bitmap* da imagem binária).

Entretanto, podemos definir uma imagem biniï¿½ria f como subconjunto X de $\mathbb{Z}^n$, de modo que

um ponto $x \in \mathbb{Z}^n$ pertence a X se, e somente se, x é um ponto do objeto de f, ou seja $f(x) = 1$.

Definição 2.1.1 *Seja f uma imagem binária em $\mathbb{Z}^n$ qualquer. Denote por X o conjunto de pontos do* foreground *da imagem binária, isto é $X = \{x : f(x) = 1\}$.*

Em algumas implementações apresentadas ao longo deste trabalho, utilizaremos o conceito de *funções indicadoras de subconjunto* [7], denotadas por $1_X(x)$, definidas como:

$$1_X(x) = \begin{cases} 0, & \text{se } x \notin X \\ 1, & \text{se } x \in X \end{cases}$$

para qualquer $x \in \mathbb{Z}^n$.

Denote x_i o valor da i-ésima coordenada (ou dimensão) de x, com i em $[1..n]$. Por exemplo, para o ponto $x \in \mathbb{Z}^3$ tal que $x = (3, 5, 1)$, temos que $x_1 = 3$, $x_2 = 5$ e $x_3 = 1$.

Em geral, uma imagem binária pode ser imersível em dois tipos de grades, a saber: grade retangular e grade hexagonal. Por simplificação, neste trabalho adotaremos a grade retangular, porém, todos os conceitos definidos nesta apresentação podem ser facilmente adaptados à grade hexagonal.

Sobre a grade retangular podemos definir o que chamamos de *relação de adjacência* entre os pontos da imagem. Uma relação de adjacência em $\mathbb{Z}^n$ é dada por uma relação $\mathbf{R}_V : \mathbb{Z}^n \times \mathbb{Z}^n$. Naturalmente, uma relação de adjacência $\mathbf{R}_V$ deve ser considerada em um espaço vetorial qualquer.

Definição 2.1.2 (conjunto de adjacência) *As relações de adjacência que interessam em MM podem ser representadas por um conjunto de vetores $V \subseteq \mathbb{Z}^n$ e definidas da seguinte maneira:*

Dados dois pontos $p, q \in \mathbb{Z}^n$, dizemos que q é **V**-adjacente *a p se, e somente se, existe um $v \in V$ tal que $q = p + v$.*

O conjunto V é chamado de conjunto de adjacência*.*

As relações de adjacência têm um papel especial. Em geral, a complexidade dos algoritmos do capítulo posterior depende intrinsecamente do tamanho do conjunto V. Tipicamente, em nossas aplicações o conjunto V é restrito a uma janela 3×3 centrada na origem.

Neste trabalho, por motivos didáticos, utilizaremos o *conjunto de vetores 4-conexo*, o *conjunto de vetores 8-conexo* e o *conjunto de vetores direcionais*.

O conjunto de **vetores 4-conexo** sobre a dimensão n, denotado por $\mathcal{V}_4$, é definido como:

$$\mathcal{V}_4 = \{p \in \{-1, 0, 1\}^n : \sum_{i=1}^{n} |p_i| \leq 1\}$$

Analogamente, o conjunto de **vetores 8-conexo** sobre a dimensão n, denotado por $\mathcal{V}_8$, é definido como:

$$\mathcal{V}_8 = \{-1, 0, 1\}^n$$

Finalmente, o conjunto de **vetores direcionais** sobre a dimensão n, denotado por $\mathcal{V}_{(1,e)}$, é definido sobre um eixo fixo $e \in [1..n]$ de $\mathbb{Z}^n$ (geralmente horizontal ou vertical) como:

$$\mathcal{V}_{(1,e)} = \{p \in \{-1, 0, 1\}^n : p_j = 0, \forall j \neq e\}$$

Usualmente, denota-se por $\mathcal{V}_{(v)}$ o conjunto de vetores direcionais orientados sobre o eixo vertical, bem como $\mathcal{V}_{(h)}$ o conjunto de vetores direcionais orientados sobre o eixo horizontal.

Vale salientar que a origem pertence a um conjunto de adjacência qualquer. Note ainda que $|\mathcal{V}_{(1,e)}| = 3$, $|\mathcal{V}_4| = 2n+1$ e $|\mathcal{V}_8| = 3^n$. Observe que o valor de n é relativamente pequeno (normalmente entre 1 e 3), o que torna a cardinalidade dos conjuntos $|\mathcal{V}_{(1,e)}|$, $|\mathcal{V}_4|$ e $|\mathcal{V}_8|$ bastante limitada. A Figura 2.1 ilustra estes conjuntos de vetores no espaço $\mathbb{Z}^2$.

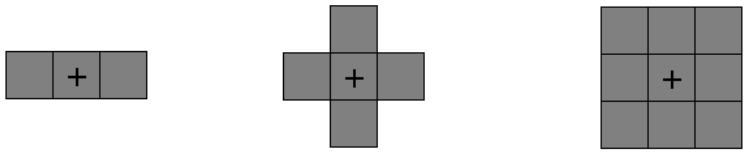


Figura 2.1: À esquerda um conjunto $\mathcal{V}_{(h)}$, ao centro o conjunto $\mathcal{V}_4$ e à direita o conjunto $\mathcal{V}_8$ no espaço $\mathbb{Z}^2$. O sinal + indica a origem.

Um vetor v de V é chamado de *anterior* se $v_i \leq 0$ para todo i em $[1..n]$ e é chamado de *posterior* se $v_i \geq 0$ para todo i em $[1..n]$.

Denote por $V^{\leftarrow}$ o conjunto dos vetores anteriores de V e por $V^{\rightarrow}$ o conjunto dos vetores posteriores de V.

Observe um exemplo destes vetores na Figura 2.2.

2.2 Morfologia Matemática

Visto que uma imagem binária pode ser representada por um subconjunto de $\mathbb{Z}^n$, vamos definir as bases matemáticas e morfológicas sobre conjuntos.

Denote por $\mathcal{P}(\mathbb{Z}^n)$ o conjunto das partes de $\mathbb{Z}^n$. Um *operador de conjunto* é qualquer mapeamento definido de $\mathcal{P}(\mathbb{Z}^n)$ em $\mathcal{P}(\mathbb{Z}^n)$.

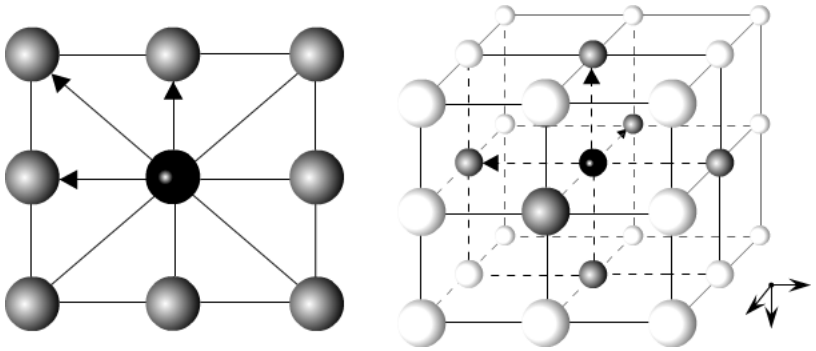


Figura 2.2: À esquerda um exemplo de 8-adjacência sobre o espaço $\mathbb{Z}^2$, e à direita um exemplo de 4-adjacência sobre o espaço $\mathbb{Z}^3$. Os pontos cinzas são adjacentes aos pontos pretos. As setas indicam adjacência anterior.

Resolver um problema de Processamento de Imagens usando MM pode ser entendido como encontrar um operador de conjunto específico $\psi : \mathcal{P}(\mathbb{Z}^n) \to \mathcal{P}(\mathbb{Z}^n)$ que seja capaz de, a partir de uma imagem de entrada $X \in \mathcal{P}(\mathbb{Z}^n)$, obter uma imagem de saída $\psi(X) \in \mathcal{P}(\mathbb{Z}^n)$.

Seja $h \in \mathbb{Z}^n$. O conjunto X_h, definido por $X_h = \{x + h : x \in X\}$, é uma *translação* de X por h. O conjunto $X^t = \{-x : x \in X\}$ é a *transposta* de X.

Um operador de conjunto ψ é dito ser *invariante por translação* (i.t.) se, e somente se, $\forall h \in \mathbb{Z}^n, \forall X \in \mathcal{P}(\mathbb{Z}^n), \psi(X_h) = \psi(X)_h$.

Sejam $X, B \in \mathcal{P}(\mathbb{Z}^n)$. As operações $X \cup B$, $X \cap B$, $X \backslash B$ e X^c São as usuais operações de união, interseção, diferença e complemento, respectivamente. O par $(\mathcal{P}(\mathbb{Z}^n), \subseteq)$ é um *reticulado Booleano completo* [10].

A operação conhecida como *adição de Minkowski* pode ser definida como:

$$A \oplus B = \bigcup_{b \in B} A_b = \{x \in \mathbb{Z}^n : \exists b \in B \text{ tal que } x - b \in A\}.$$

A *subtração de Minkowski* pode ser definida como:

$$A \ominus B = \bigcap_{b \in B} A_{-b} = \{x \in \mathbb{Z}^n : \forall b \in B \text{ tal que } x + b \in A\}.$$

Existem outras maneiras de se definir as operações de *Minkowski* [1], todavia opta-se neste trabalho pelas definições anteriores, visto que são as definições mais usuais.

[1] Por exemplo, Pierre Soille [38] em seus trabalhos define que $A \ominus B = \bigcap_{b \in B} A_b$.

Definição 2.2.1 (adjacência) *Observe que a partir da* adição de Minkowski *podemos, alternativamente, definir a relação de adjacência entre dois pontos $p, q \in \mathbb{Z}^n$ sobre o conjunto de adjacência V do seguinte modo:*

$$\textit{Dizemos que } q \textit{ é } \mathbf{V}\textit{-adjacente a } p \textit{ se, e somente se, } q \in \{p\} \oplus V.$$

Conseqüentemente, $p \in \{q\} \oplus V^t$.

Observe que se q é **V**-adjacente a p, **não** necessariamente p é **V**-adjacente a q. Por exemplo, considere V_{a} e V_{b} da Figura 2.3. Esta figura nos mostra um exemplo em que o ponto p é V_{a}-adjacente a q, mas q não é V_{a}-adjacente a p. No entanto, p é V_{b}-adjacente a q, bem como q é V_{b}-adjacente a p. A recíproca só é contemplada no caso em que o conjunto V é *simétrico*.

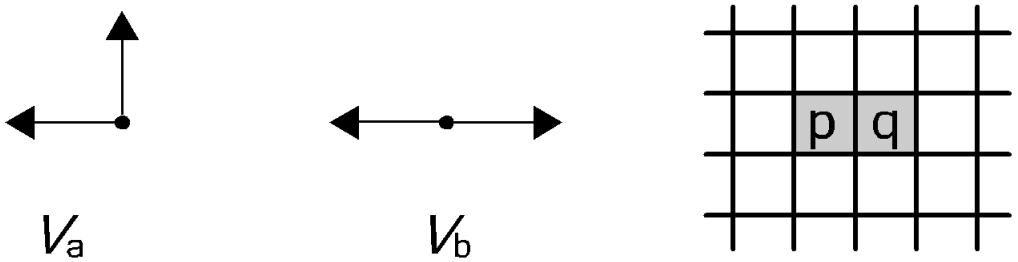


Figura 2.3: Neste exemplo o ponto p é V_{a}-adjacente a q, mas q não é V_{a}-adjacente a p. Como V_{b} é simétrico, a reciprocidade é válida.

2.3 Dilatação e Erosão

Vamos definir as tão citadas erosão e dilatação. Sejam $X, Y, B \in \mathcal{P}(\mathbb{Z}^n)$. Os operadores de conjunto i.t. δ_B e ε_B são, respectivamente, a dilatação e a erosão por B definidas por:

$$\delta_B(X) = X \oplus B \text{ e } \varepsilon_B(X) = X \ominus B.$$

O conjunto B é chamado de *elemento estruturante*.

Propriedade 2.3.1 (propriedades da erosão e dilatação) *Existem inúmeras propriedades da erosão e da dilatação. Citemos, pois, algumas de maior importância para este trabalho:*

1. *Dilatação e Erosão são operadores crescentes (isotônicos), isto é, se $X \subseteq Y$, então $\delta_B(X) \subseteq \delta_B(Y)$ e $\varepsilon_B(X) \subseteq \varepsilon_B(Y)$.*

2. *Dilatação é comutativa, ou seja,* $\delta_B(X) = \delta_X(B)$.

3. *Dilatação é associativa, ou seja,* $\delta_B(\delta_Y(X)) = \delta_{\delta_B(Y)}(X)$.

4. *Dilatação e Erosão são operadores i.t., isto é, para um* $h \in \mathbb{Z}^n$ *qualquer, valem* $\delta_B(X_h) = (\delta_B(X))_h$ *e* $\varepsilon_B(X_h) = (\varepsilon_B(X))_h$.

5. *Valem também* $\delta_{B_h}(X) = (\delta_B(X))_h$ *e* $\varepsilon_{B_h}(X) = (\varepsilon_B(X))_{-h}$.

6. *Dilatação e Erosão são operadores duais:* $\delta_B(X) = (\varepsilon_{B^t}(X^c))^c$ *e* $\varepsilon_B(X) = (\delta_{B^t}(X^c))^c$.

7. *Dilatação e Erosão são operadores distributivos, respectivamente, quanto à união e interseção, isto é,* $\delta_B(\cup_{i=1}^n X_i) = \cup_{i=1}^n \delta_B(X_i)$ *e* $\varepsilon_B(\cap_{i=1}^n X_i) = \cap_{i=1}^n \varepsilon_B(X_i)$.

8. *Dilatação é duplamente distributiva quanto à união:* $\delta_{\cup_{j=1}^m B_j}(\cup_{i=1}^n X_i) = \cup_{i=1}^n \cup_{j=1}^m \delta_{B_j}(X_i)$.

9. *Para Erosão vale que* $\varepsilon_{\cup_{i=1}^n B_i}(X) = \cap_{i=1}^n \varepsilon_{B_i}(X)$.

10. *Se a origem está em* B, *então* $\varepsilon_B(X) \subseteq X \subseteq \delta_B(X)$.

11. *Seqüência de Dilatações e Erosões por elementos estruturantes* $B_1, B_2, \ldots, B_n$ *podem ser reduzidas, isto é,* $\delta_{B_1}(\delta_{B_2}(\ldots(\delta_{B_n}(X))\ldots)) = \delta_{B_1 \oplus B_2 \oplus \ldots \oplus B_n}(X)$ *e* $\varepsilon_{B_1}(\varepsilon_{B_2}(\ldots(\varepsilon_{B_n}(X))\ldots)) = \varepsilon_{B_1 \oplus B_2 \oplus \ldots \oplus B_n}(X)$.

Tais propriedades provêm das propriedades de adição e subtração de *Minkowski* [20]. A partir da combinação de várias destas propriedades, conseguimos obter métodos mais rápidos do que aplicar diretamente a dilatação ou a erosão da definição. Por exemplo, desta última propriedade surgiram vários métodos de decomposição de elemento estruturante (ver Cap. 4).

Existe também uma definição alternativa para dilatação e erosão, proveniente da interpretação geométrica destas operações:

$$\delta_B(X) = \{h : X \cap B_h^t \neq \emptyset\} \text{ e}$$

$$\varepsilon_B(X) = \{h : B_h \subseteq X\}.$$

Observe uma interpretação geométrica de dilatação e erosão sobre uma imagem bidimensional na Figura 2.4.

Além disso, como a dilatação e a erosão são operadores duais, a implementação eficiente de um destes operadores pode ser reduzida ao outro operador. Tal implementação eficiente também pode refletir uma eficiente implementação da composição de outros operadores morfológicos.

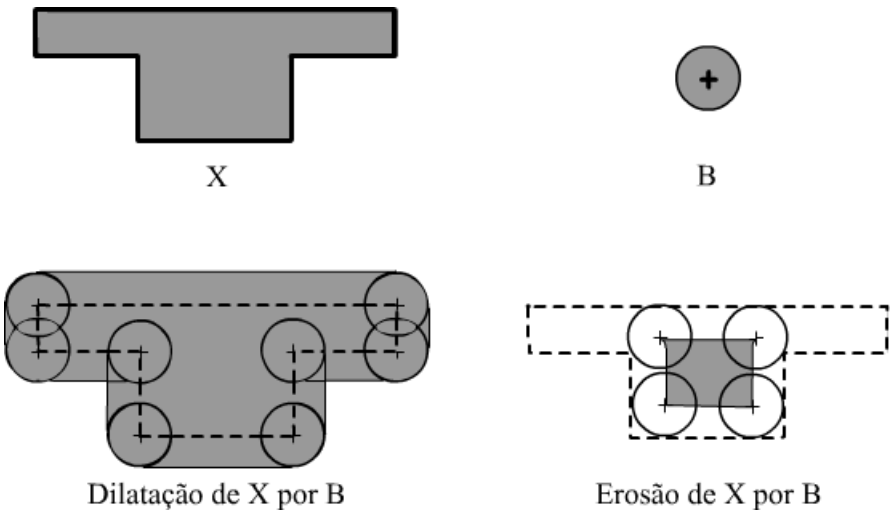


Figura 2.4: Um exemplo de dilatação e erosão de X por B. O sinal + em B indica a origem.

2.4 Exemplos de Outros Operadores Morfológicos

Vamos nesta seção citar alguns dos operadores mais comuns na MM, que podem ser decompostos em termos de dilatação e erosão.

O operador de *abertura morfológica* de uma imagem $X \in \mathcal{P}(\mathbb{Z}^n)$ por um elemento estruturante $B \in \mathcal{P}(\mathbb{Z}^n)$, denotado por $\gamma_B(X)$, é definido como:

$$\gamma_B(X) = X \circ B = \delta_B(\varepsilon_B(X)) = \bigcup_{h \in \mathbb{Z}^n} \{B_h : B_h \subseteq X\}.$$

A abertura é bastante importante para a MM, e existem até algoritmos dedicados à tal operação [43].

O operador de *fechamento morfológico* de X por B, denotado por $\phi_B(X)$, é definido como:

$$\phi_B(X) = X \bullet B = \varepsilon_B(\delta_B(X)).$$

Note que tanto o operador de abertura quanto o de fechamento são idempotentes, ou seja:

$$\gamma_B(X) = \gamma_B(\gamma_B(X)) \text{ e } \phi_B(X) = \phi_B(\phi_B(X)).$$

A combinação destes operadores é bastante usada em eliminação de ruídos. Observe um exemplo de abertura e fechamento na Figura 2.5.

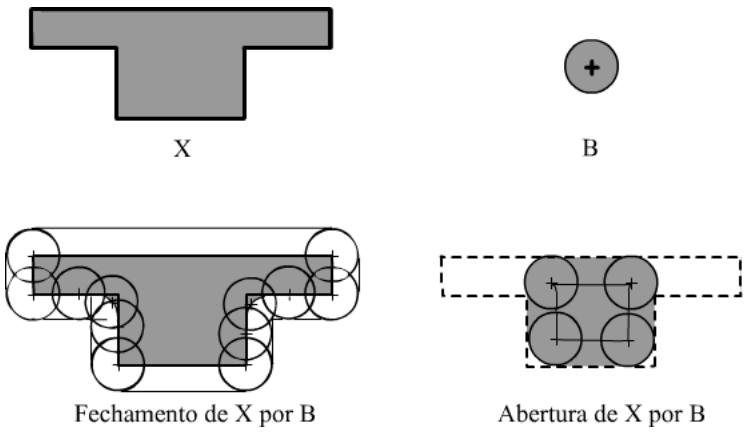


Figura 2.5: Um exemplo de abertura e fechamento de X por B.

Existe ainda uma classe especial de operadores: os *W-operadores*. Sabe-se que um operador ψ é localmente definido (l.d.) em uma janela $W \subseteq \mathbb{Z}^n$ se, e somente se, para todo $X \in \mathcal{P}(\mathbb{Z}^n)$ e $\forall h \in \mathbb{Z}^n, h \in \psi(X) \Leftrightarrow h \in (X \cap W_h)$. Um operador i.t. e l.d. em W é chamado de *W-operador*.

O W-operador *sup-gerador* (ou *hit-or-miss*) de uma imagem $X \in \mathcal{P}(\mathbb{Z}^n)$ pelos elementos estruturantes $A, B \in \mathcal{P}(\mathbb{Z}^n)$, denotado por $\lambda^W_{A,B}(X)$, é definido como:

$$\lambda^W_{A,B}(X) = \{h \in \mathbb{Z}^n : A_h \subseteq X \cap W_h \subseteq B_h\},$$

onde $W \subseteq \mathbb{Z}^n$ com $A \subseteq B \subseteq W$.

Uma propriedade do operador sup-gerador é que ele pode ser descrito como uma interseção de uma erosão e uma dilatação:

$$\lambda^W_{A,B}(X) = \varepsilon_A(X) \cap (\delta_{(B^c)^t}(X))^c.$$

E pela dualidade da dilatação:

$$\lambda^W_{A,B}(X) = \varepsilon_A(X) \cap \varepsilon_{B^c}(X^c).$$

O conjunto B^c é restrito ao tamanho da janela W. Sabe-se que qualquer operador i.t. pode ser basicamente decomposto em termos de dilatação e erosão [6, 7].

Enfim, uma implementação rápida da erosão e dilatação pode levar a uma implementação rápida do operador sup-gerador.

Sendo assim, vamos apresentar uma nova abordagem para implementações rápidas de erosão e dilatação binária no capítulo seguinte.

Capítulo 3

Técnicas de Pré-Processamento

Antes de adentrarmos as discussões sobre os novos métodos para dilatação e erosão morfológica binária, é mister que se fundamente alguns conceitos que justifiquem a boa performance destes algoritmos.

A motivação em pré-processar os conjuntos que representam as imagens binárias fora impulsionada por estudos sobre técnicas de pré-processamento de padrões e textos, utilizadas em buscas de strings (*string-matching*). Tais técnicas, de um modo geral, possibilitam obter uma representação mais compacta dos conjuntos pré-processados. A partir desta representação compacta, este trabalho propõe desenvolver alguns algoritmos para calcular mais rapidamente a dilatação e a erosão morfológica binária. Espera-se que os algoritmos de pré-processamento consumam tempo linear (ou quase linear) no tamanho da entrada. Como exemplo, é possível extrair o contorno de um conjunto em tempo linear e a partir deste, calcular mais eficientemente as operações morfológicas.

Grande parte das definições e conceitos deste capítulo foram criadas juntamente com o *prof. Dr. Ronaldo Fumio Hashimoto* e o *prof. Dr. Alair Pereira Lago*.

Dentre as técnicas de pré-processamento apresentadas por este trabalho estão o *Extrator de Contorno Direcional* (Seção 3.1) e a *Transformada da Densidade* (Seção 3.2).

3.1 Pré-Processamento de Contornos

Nesta seção, citaremos apenas alguns conceitos relativos à extração do contorno, visto que é um problema bem explorado pela literatura clássica [3, 4, 24]. Entretanto, este trabalho propõe um novo conceito conhecido como *contorno direcional*.

Definição 3.1.1 (contorno direcional) *Sejam $X \subseteq \mathbb{Z}^n$ e V um conjunto de adjacência.*

O **V**-contorno *do conjunto X, denominado contorno direcional de X, é o conjunto:*

$$cont^V(X) = \{p \in X : \exists q \notin X \text{ tal que } q \text{ é } \mathbf{V}\text{-adjacente a } p\}.$$

Existem várias maneiras de extrair um contorno direcional de um conjunto X. Este trabalho propõe um algoritmo **Contour**(X, V) para extração de contorno direcional em tempo $O(|X| \cdot |V|)$ [1], onde o conjunto V utilizado pelo algoritmo é um conjunto de adjacência.

Algoritmo 1:Contour(X, V) **{**

```
1:      cont^V(X) ← ∅;
2:      Para todo x ∈ X faça {
3:          Para todo v ∈ V faça
4:              Se (x + v ∉ X) então
5:                  cont^V(X) ← cont^V(X) ∪ {x};
        }
6:      Devolva cont^V(X);
```

}

3.1.1 Análise de Complexidade

Em geral, a estrutura de dados usada para armazenar o conjunto V e o conjunto $cont^V(X)$ no algoritmo é uma *lista ligada*, e para a imagem binária X uma *matriz multidimensional*. Para a análise o consumo de tempo do algoritmo Contour, considere a seguinte tabela:

Linha do Algoritmo	Consumo de Tempo
1	$\Theta(1)$
2	$\Theta(\|X\|)$
3	$\Theta(\|X\| \cdot \|V\|)$
4	$\Theta(\|X\| \cdot \|V\|)$
5	$O(\|X\| \cdot \|V\|)$
6	$\Theta(1)$
Total	$\Theta(\|X\| \cdot \|V\|)$

Podemos extrair o contorno direcional de qualquer subconjunto de $\mathbb{Z}^n$, para um n qualquer. Considerando que $|V|$ é constante, podemos dizer que o algoritmo Contour consome tempo linear.

[1] Em todos os algoritmos propostos por este trabalho, considere que a busca por um elemento dentro de um conjunto qualquer é realizada em tempo constante.

Todavia, devido a flexibilidade do algoritmo quanto ao tamanho de V, dizemos que este consome tempo *quase linear*.

Proposição 3.1.2 *Existe ainda, uma propriedade interessante que nos mostra que:*

$$cont^V(X) = X \setminus (X \ominus V).$$

Prova: *Pela Definição 3.1.1,* $cont^V(X) = \{p \in X : \exists q \notin X$ *tal que* q *é* $\mathbf{V}$*-adjacente a* $p\}$. *Além disso, aplicando a Definição 2.2.1 de adjacência temos que* $cont^V(X) = \{p \in X : \exists q \notin X$ *tal que* $p \in \{q\} \oplus V^t\}$. *Ou seja, uma vez que* $o \in V^t$ *e* $q \in X^c$, *temos que* $cont^V(X) = \{p \in X : p \in X^c \oplus V^t\} = \{p \in X : p \in \delta_{V^t}(X^c)\}$. *Pela dualidade da erosão/dilatação (ver Seção 2.3) temos que* $cont^V(X) = \{p \in X : p \in (\varepsilon_V(X))^c\} = \{p \in X : p \notin \varepsilon_V(X)\} = \{p \in X : p \notin X \ominus V\}$. *Visto que* $p \in X$ *e* $p \notin X \ominus V$, *temos que* $cont^V(X) = X \setminus (X \ominus V)$. $\diamond$

A Figura 3.1 exemplifica o V-contorno de um conjunto X. Nesta figura os pontos *foreground* estão representados pelos pontos em cinza. Observe que o conjunto V é composto por apenas um vetor orientado para cima. Neste caso, os pontos pertencentes a este contorno direcional são os extremos superiores de segmentos verticais maximais presentes em X.

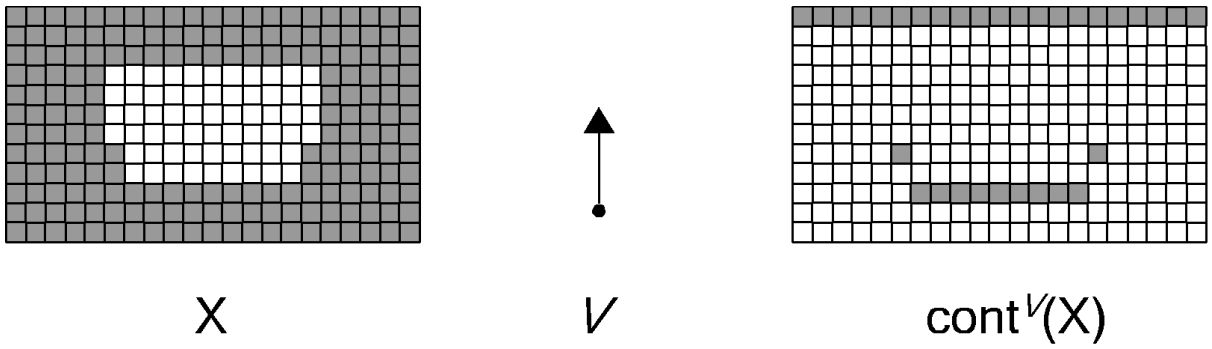


Figura 3.1: À esquerda o conjunto X, ao centro o conjunto V e à direita o conjunto $cont^V(X)$.

Fundamentalmente, utilizaremos o extrator de contorno direcional como subrotina da Transformada da Densidade, conforme veremos na seção seguinte.

3.2 Transformada da Densidade

A *Transformada da Densidade* se assemelha muito à Transformada da Distância [12]. Na Transformada da Densidade, de alguma forma a densidade de um ponto corresponde a uma certa "distância" deste ponto em relação a borda da imagem que o contém, de acordo com um conjunto de adjacência.

Os pesquisadores Chen e Haralick [39] desenvolveram a Transformada da Erosão Recursiva, bastante semelhante a transformada da densidade.

Ela foi desenvolvida para diminuir o espaço de busca de um elemento estruturante dentro da imagem de entrada (no caso da erosão), ou simplesmente, criar uma representação compacta (ver adiante a codificação por cascas) dos conjuntos para aplicação das operações morfológicas (no caso da dilatação). Espera-se, futuramente, utilizar a Transformada da Densidade em muitas outras aplicações dentro da MM, tais como a granulometria, análise de textura, abertura morfológica, supgeradores, análises de sinais, compactação de imagens binárias, ...

Para se ter uma intuição sobre o funcionamento da Transformada da Densidade, imagine a seguinte ilustração: "*Em uma determinada mata fechada, um grupo de agricultores realiza uma queimada coordenada. Focos são acesos ao mesmo tempo em todo perímetro da mata. É de se imaginar que, a medida que o fogo avança em direção ao centro da mata, este não retrocede, consumindo regiões da mata que já foram consumidas anteriormente. Outro fato interessante é que cada ponto consumido pelo fogo é um novo foco de queimada. Considerando que o fogo se propaga com velocidade uniforme, a densidade de um ponto qualquer dentro da mata pode ser compreedida como o deslocamento de espaço desde o início da queimada até o momento em que fogo atinge tal ponto. O conjunto a ser processado pode ser visto como a própria mata*".

Para entender melhor tal transformada é necessário que se entenda os seguintes conceitos:

Definição 3.2.1 (caminho direcional) *Seja V um conjunto de adjacência. A seqüência de pontos $P = (p = p_1, p_2, \ldots, p_m = q)$ é um* $\mathbf{V}$-caminho *que* liga *p a q se, e somente se, para todo par de pontos distintos p_i e p_{i+1} temos que p_{i+1} é $\mathbf{V}$-adjacente a p_i, $\forall i \in [1..m-1]$.*

O comprimento do $\mathbf{V}$-caminho P é definido como $|P| - 1$.

Definição 3.2.2 (densidade entre dois pontos) *Sejam $p, q \in \mathbb{Z}^n$ e V um conjunto de adjacência qualquer. A* $\mathbf{V}$-densidade *de um ponto p a um ponto q, denotado por $d^V(p,q)$, é o comprimento do menor $\mathbf{V}$-caminho que liga p a q.*

Caso tal $\mathbf{V}$-caminho não exista, dizemos que $d^V(p,q) = +\infty$.

Dados três pontos quaisquer $p, q, r \in \mathbb{Z}^n$. Seguem algumas propriedades sobre tais pontos:

- Define-se que $d^V(p,p) = 0$, para qualquer V.
- Para qualquer conjunto simétrico de vetores de adjacência S temos que $d^S(p,q) = d^S(q,p)$.
- Vale a desigualdade triangular $d^V(p,r) \leq d^V(p,q) + d^V(q,r)$.
- Vale que $d^V(p,q) \geq 0$, para qualquer V.

Segundo algumas literaturas clássicas [22], podemos considerar que a V-densidade de p a q é uma *distância* somente se o conjunto de adjacência V é simétrico.

Ao escolhermos o conjunto $\mathcal{V}_4$ como o conjunto de adjacência, a $\mathcal{V}_4$-densidade $d^{\mathcal{V}_4}(p,q)$ é a bem conhecida distância *cityblock* de p a q, assim como ao escolhermos o conjunto $\mathcal{V}_8$, $d^{\mathcal{V}_8}(p,q)$ passa a ser a distância *manhattan* [12]. A Figura 3.2 mostra um exemplo destas distância entre dois pontos p e q. Neste caso, a distância *cityblock* entre p e q é 15 e a a distância *manhattan* entre p e q é 11.

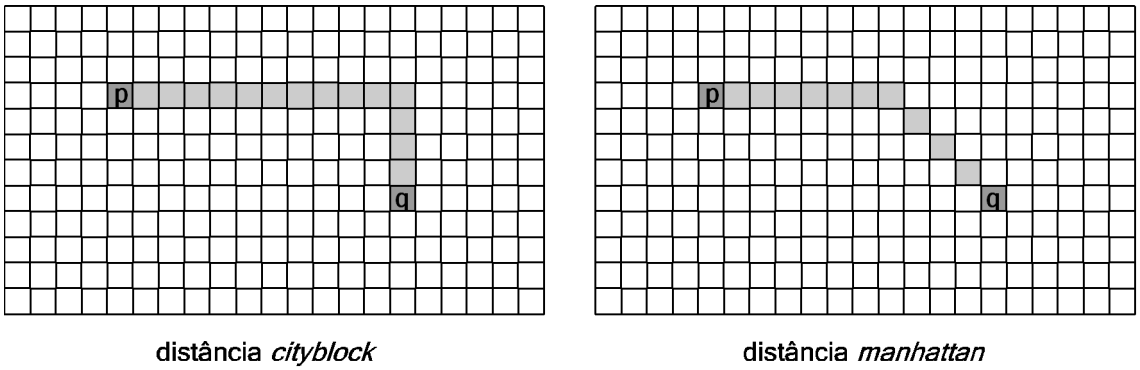


Figura 3.2: À esquerda a distância *cityblock* entre p e q e à direita a distância *manhattan* entre estes mesmos pontos. Note que $d^{\mathcal{V}_4}(p,q) = 15$ e $d^{\mathcal{V}_8}(p,q) = 11$.

Definição 3.2.3 (densidade absoluta) *Sejam $X \subseteq \mathbb{Z}^n$, $p \in \mathbb{Z}^n$ e V um conjunto de adjacência qualquer.*

(a) A **V**-densidade *de um ponto p em relação a X, denotado por $\Delta_X^V(p)$, é definido como a menor densidade $d^V(p,q), \forall q \in X^c$.*

(b) Alternativamente, a **V**-*densidade $\Delta_X^V(p)$ pode ser definida pela seguinte recorrência:*

$$\Delta_X^V(p) = \begin{cases} 0, & \text{se } p \notin X \\ \min_{\forall v \in V}\{\Delta_X^V(p+v)\} + 1, & \text{caso contrário.} \end{cases}$$

Se para cada ponto calculássemos a menor densidade $d^V(p,q), \forall q \in X^c$, a Transformada da Densidade seria muito onerosa, chegando a ordem $O(|X|^2)$. Por outro lado, a implementação direta da definição recursiva poderia ocasionar erros lógicos na Transformada da Densidade. Por exemplo, calcular a $\mathcal{V}_4$-densidade de um ponto centrado em um quadrado 5×5 levará o algoritmo a entrar em um laço infinito, visto que o conjunto $\mathcal{V}_4$ é simétrico.

Para resolver tais problemas, opta-se realizar a Transformada da Densidade a partir dos pontos do conjunto que contém menores densidades, e a partir destes, calcular incrementalmente as den-

sidades dos outros pontos. O princípio adotado nesta abordagem é o princípio da *propagação de fronteiras* [28], conforme fora sugerido na ilustração no início desta seção.

Proposição 3.2.4 *Sejam $X \subseteq \mathbb{Z}^n$ e V um conjunto de adjacência. Os pontos do V-contorno de X são pontos p para os quais existe um $v \in V$ tal que $q = p + v$ não pertence a X , e pela definição de* **V***-densidade, a densidade $\Delta_X^V(p) = 1$. Sendo assim, temos que todo ponto p de X tem V-densidade 1 em relação a X se, e somente se, p pertence ao V-contorno de X.*

Prova: *Um ponto p pertence ao V-contorno de X se, e somente se, $p \in X$ e $\exists q \notin X$ tal que q é V-adjacente a p, ou seja, $\exists u \in V$ tal que $q = p + u$. Como $q \notin X$, temos que $\Delta_X^V(q) = 0$ e conseqüentemente $\Delta_X^V(p + u) = 0$. Logo, pela definição de* **V***-densidade, como $p \in X$, $\Delta_X^V(p) = \min_{\forall v \in V}\{\Delta_X^V(p + v)\} + 1$. Uma vez que $q = p + u$ para algum $u \in V$ e $\Delta_X^V(q) = 0$, então $\min_{\forall v \in V}\{\Delta_X^V(p + v)\} = 0$. Logo $\Delta_X^V(p) = 1$.* $\diamond$

A partir dos pontos de V-densidade 1, o próximo passo é encontrar todos os pontos de V-densidade 2.

Denote por $front_i^V(X)$ o subconjunto dos pontos de X cuja V-densidade é i, para $i > 0$. Conforme observado anteriormente, $front_1^V(X) = cont^V(X)$. Note que podemos encontrar os pontos de V-densidade 2, ou seja $front_2^V(X)$, do mesmo modo em que encontramos os pontos de $front_1^V(X)$. O conjunto $front_2^V(X)$ é o V-contorno do conjunto $X \setminus front_1^V(X) = X \setminus cont^V(X)$.

Proposição 3.2.5 *De um modo geral, $front_i^V(X) = cont^V(X \setminus (\cup_{1 \leq j \leq i-1} front_j^V(X)))$.*

Prova: *Conforme fora dito, o conjunto $front_i^V(X)$ é o conjunto dos pontos $p \in X$ tais que $\Delta_X^V(p) = i$. Entretanto, sabe-se que o conjunto $\cup_{1 \leq j \leq i-1} front_j^V(X)$ é o conjunto de todos os pontos $q \in X$ tais que $\Delta_X^V(q) \leq i - 1$. Tomando-se todos os pontos $p' \in cont^V(X \setminus (\cup_{1 \leq j \leq i-1} front_j^V(X)))$ temos que $\Delta_X^V(p') = \min\{\Delta_X^V(q) : \forall q \in \cup_{1 \leq j \leq i-1} front_j^V(X)$ e q é V-adjacente a $p'\} + 1$. Certamente $\Delta_X^V(p') \leq i$, já que $\Delta_X^V(q) \leq i - 1$. Além disso, como $\Delta_X^V(p') > i - 1$, uma vez que $p' \notin \cup_{1 \leq j \leq i-1} front_j^V(X)$, temos que $\Delta_X^V(p') = i$. Além do mais, não é difícil mostrar que para todo $r \in X$ tal que $r \notin cont^V(X \setminus (\cup_{1 \leq j \leq i-1} front_j^V(X)))$ vale que $\Delta_X^V(r) > i$. Portanto, o conjunto $cont^V(X \setminus (\cup_{1 \leq j \leq i-1} front_j^V(X)))$ é o conjunto dos pontos $p' \in X$ tais que $\Delta_X^V(p') = i$, ou seja, $front_i^V(X) = cont^V(X \setminus (\cup_{1 \leq j \leq i-1} front_j^V(X)))$.* $\diamond$

Note que, necessariamente, a V-densidade de um ponto $p \in front_i^V(X)$ é i, já que o ponto q V-adjacente a p de menor V-densidade tem $\Delta_X^V(q) = i - 1$. Esta idéia nos sugere "propagar" as **V**-densidades a partir de cada contorno $front_i^V(X)$. Cada $front_i^V(X)$ é chamado de *fronteira* ou *frente de propagação*.

Sendo assim, podemos obter um $front_i^V(X)$ a partir de $front_{i-1}^V(X)$, conforme mostramos no esquema a seguir:

$$front_i^V(X) = \begin{cases} cont^V(X), & \text{se } i = 1 \\ ((front_{i-1}^V(X) \oplus V^t) \setminus front_{i-1}^V(X)) \cap X, & \text{caso contrário.} \end{cases}$$

Esta propagação é basicamente uma otimização da operação $cont^V(X \setminus (\cup_{1 \leq j \leq i-1} front_j^V(X)))$. A Figura 3.3 mostra um exemplo de frente de propagação já na terceira iteração. Neste exemplo, todos os pontos de $front_3^V(X)$ têm V-densidade 3 em relação a X.

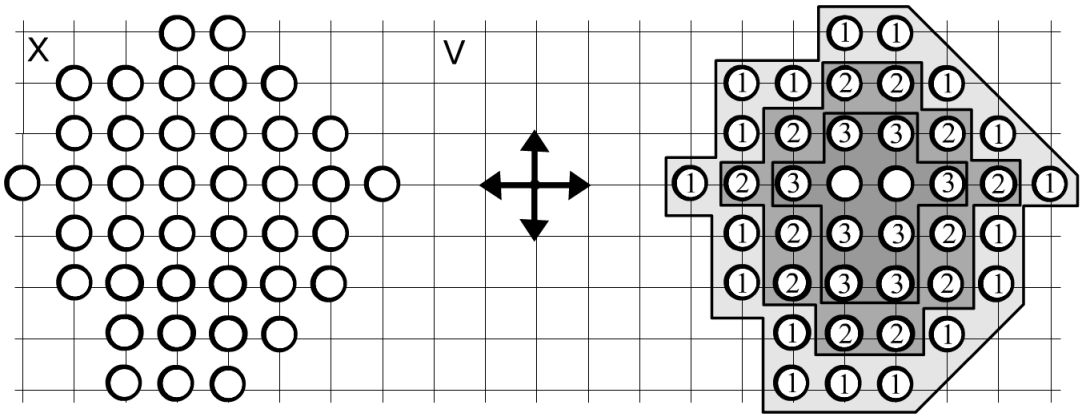


Figura 3.3: À esquerda o conjunto X, ao centro o conjunto de adjacência V e à direita a terceira iteração da frente de propagação na Transformada da Densidade.

Podemos inicialmente encontrar todos os pontos com densidade 1, e a partir destes, encontrar incrementalmente os de densidade superior, utilizando o mesmo princípio de propagação de fronteiras, até visitar todos os pontos de X.

Segue um algoritmo **setDensity**(X, V) que calcula as **V**-densidades de todos os pontos de um conjunto X a partir do **V**-contorno do conjunto X, utilizando a técnica de propagação de fronteiras.

Algoritmo 2: setDensity(X, V) {

```
1:      i ← 1;
2:      front_i ← cont^V(X);
3:      Enquanto front_i ≠ ∅ faça {
4:          Para todo x ∈ front_i faça
5:              Δ^V_X(x) ← i;
6:          i ← i + 1;
7:          front_i ← ((front_{i-1} ⊕ V^t) \ front_{i-1}) ∩ X;
        }
}
```

3.2.1 Análise de Complexidade

Seja $i_{\max}$ o maior índice i no algoritmo anterior. Assim como no algoritmo Contour, para a análise o consumo de tempo do algoritmo setDensity, considere a seguinte tabela:

Linha do Algoritmo	Consumo de Tempo
1	$\Theta(1)$
2	$\Theta(\|X\| \cdot \|V\|)$
3	$\Theta(i_{\max})$
4	$\Theta(i_{\max})$
5	$\sum_{i=1}^{i_{\max}} \Theta(\|front_i\|) = \Theta(\|X\|)$
6	$\Theta(i_{\max})$
7	$\sum_{i=1}^{i_{\max}} \Theta(\|front_i\| \cdot \|V\|) = \Theta(\|X\| \cdot \|V\|)$
Total	$\Theta(\|X\| \cdot \|V\|)$

A complexidade computacional do algoritmo setDensity é $\Theta(|X| \cdot |V|)$, visto que a Linha 7 do algoritmo consome tempo $\Theta(|front_i| \cdot |V|)$ para cada i, e que o somatório de todos os conjuntos $front_i$ equivale a X. Considerando que $|V|$ é constante, o algoritmo setDensity consome tempo linear no tamanho de X para calcular as **V**-densidades de todos os pontos de X.

Pode-se também utilizar uma técnica de programação dinâmica conhecida como *memoized table* [2] [30] para realizar tal propagação.

Nos demais algoritmos, adote por simplificação, que as **V**-densidades dos pontos $x \in \mathbb{Z}^n$ usados são valores previamente calculados pelo algoritmo setDensity se $x \in X$; ou simplesmente 0, se o valor de x não foi calculado pelo mesmo ($x \notin X$).

Observe um exemplo de $\mathcal{V}^{\leftarrow}_{(1,e)}$-densidade, também chamada de *densidade direcional anterior*, avaliada na coordenada vertical e horizontal, um exemplo de $\mathcal{V}^{\leftarrow}_{4}$-densidade, conhecida por *densidade 4-conexa anterior*, e um exemplo de $\mathcal{V}^{\leftarrow}_{8}$-densidade, ou *densidade 8-conexa anterior*, todas calculadas

[2] Técnica na qual se utiliza uma tabela que armazena rótulos já calculados.

sobre o mesmo conjunto, conforme mostrado na Figura 3.4.

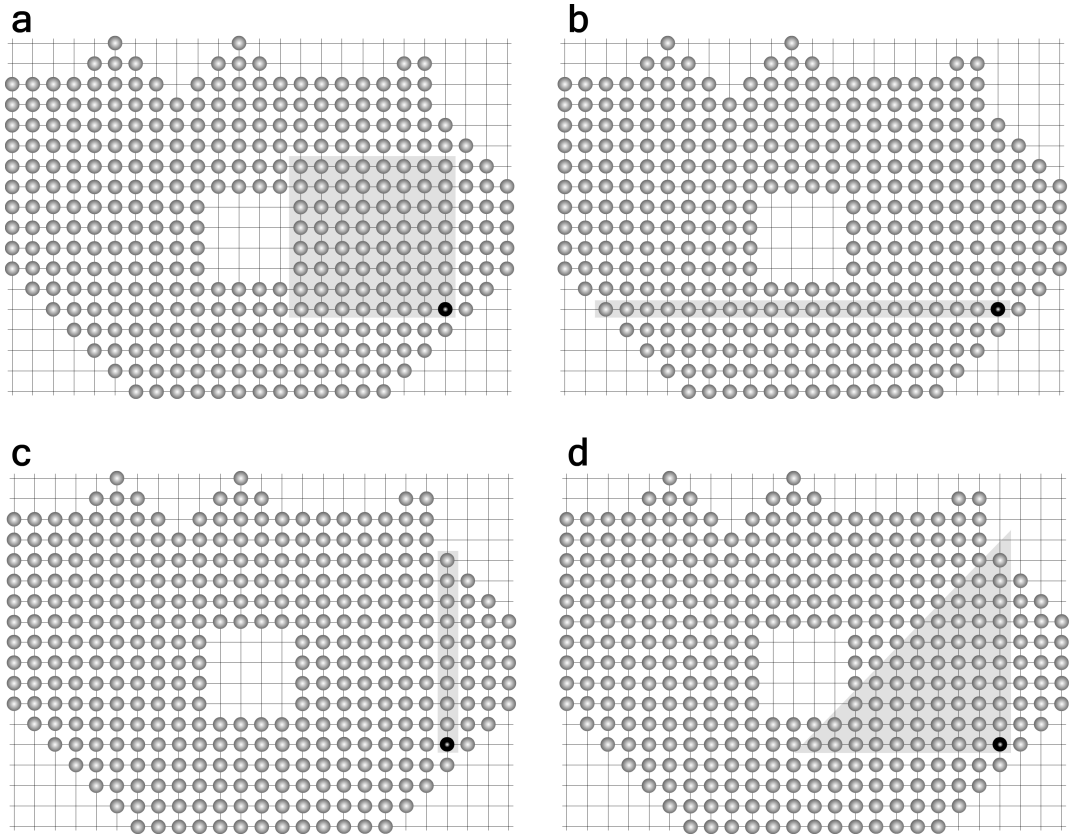


Figura 3.4: (a) a $\mathcal{V}_8^{\leftarrow}$-densidade, (b) a $\mathcal{V}_{(h)}^{\leftarrow}$-densidade e (c) a $\mathcal{V}_{(v)}^{\leftarrow}$-densidade e a (d) $\mathcal{V}_4^{\leftarrow}$-densidade do ponto preto com relação à imagem X é 8 e 16, e 10 e 10, respectivamente.

Nesta figura, o ponto preto tem sua **V**-densidade de acordo com a adjacência definida sobre o conjunto X. Por exemplo, se tomarmos o conjunto $\mathcal{V}_{(h)}^{\leftarrow}$ no sentido horizontal, a **V**-densidade do ponto preto equivale ao número de pontos anteriores horizontalmente ao ponto preto, até encontrar um ponto do fundo da imagem, e que neste caso é 16. Ao escolher o mesmo conjunto $\mathcal{V}_{(v)}^{\leftarrow}$ no sentido vertical, a **V**-densidade do ponto preto em relação a X passa a ser 10. Se escolhermos ao invés do conjunto $\mathcal{V}_{(v)}^{\leftarrow}$, o conjunto $\mathcal{V}_4^{\leftarrow}$, a **V**-densidade do ponto preto com relação a X continua a ser 10. No entanto, ao escolhermos o conjunto $\mathcal{V}_8^{\leftarrow}$ para definir a adjacência, a **V**-densidade do ponto preto em relação a X passa a ser 8.

Esta relação entre um ponto de X e os outros pontos "anteriores" a este define o que denominamos *relação de cobertura*.

Definição 3.2.6 (cobertura) *Sejam $X, V \subseteq \mathbb{Z}^n$ e $x \in \mathbb{Z}^n$. O conjunto $COB_X^V(x) \subseteq X$ é o conjunto de pontos* **V**-cobertos *por x em X, definido como:*

$$COB_X^V(x) = \{q \in X : d^V(x, q) < \Delta_X^V(x)\}$$

Esta última definição introduz o conceito de *cobertura* entre pontos de um conjunto $X \subseteq \mathbb{Z}^n$ de acordo com conjunto de adjacência V: Para algum ponto $x \in \mathbb{Z}^n$, dizemos um ponto $q \in \mathbb{Z}^n$ é **V**-*coberto* por x em X (ou x **V**-*cobre* q em X) se, e somente se, $q \in COB_X^V(x)$.

Definição 3.2.7 (dominância) *Sejam $X, Y, V \subseteq \mathbb{Z}^n$. Quando $COB_Y^V(q) \subseteq COB_X^V(x)$, além de x* **V**-*cobrir q em X, dizemos que x em X* **V**-domina *q em Y, ou ainda que q em Y é* **V**-dominado *por x em X. Quando o contexto do domínio ocorre em um único conjunto, isto é $X = Y$, dizemos simplesmente que x* **V**-*domina q.*

Ou seja, se $COB_X^V(q) \subseteq COB_X^V(x)$, significa que o ponto x é mais "representativo" do que o ponto q em X. Com base no conceito de cobertura, podemos definir o conjunto de cascas de X.

3.3 Conjunto de Cascas

Conforme dito anteriormente, um conjunto de cascas corresponde a uma representação compacta de um conjunto qualquer. Para se ter uma intuição, o conjunto de cascas de um conjunto X é basicamente o *esqueleto* de X [29].

Definição 3.3.1 (cascas) *Sejam $X \subseteq \mathbb{Z}^n$, $\zeta \in X$ e V um conjunto de adjacência.*

(a) Se o ponto ζ não pode ser **V**-dominado *por nenhum outro ponto em X, dizemos que o ponto ζ é uma* **V**-casca *de X.*

(b) Alternativamente, ζ é uma **V**-casca *de X se, e somente se, para todo $v \in V^t$ temos que $\Delta_X^V(\zeta + v) \leq \Delta_X^V(x)$.*

A definição alternativa (b) de **V**-casca se baseia no fato de que se para algum $v \in V^t$ vale que $\Delta_X^V(x + v) > \Delta_X^V(x)$, então x certamente não é uma **V**-casca (ver Prop. 3.3.3).

Geometricamente, uma *casca direcional horizontal anterior*, ou seja, uma $\mathcal{V}_{(h)}^{\leftarrow}$-casca pode ser interpretada como o ponto extremo mais à direita do segmento maximal que a contém, dentro da imagem binária. Analogamente, uma *casca 8-conexa anterior* (ou $\mathcal{V}_8^{\leftarrow}$-casca) é o vértice inferior direito de um quadrado maximal que contém tal casca no espaço $\mathbb{Z}^2$.

Observe o exemplo da Figura 3.5, que é uma imagem que contém 337 pontos. Os números indicados no centro dos pontos correspondem às $\mathcal{V}_8^{\leftarrow}$-densidades destes pontos. Observe que cada ponto com $\mathcal{V}_8^{\leftarrow}$-densidade m é o extremo inferior direito de um quadrado maximal $m \times m$ dentro da imagem binária. Se tomarmos as $\mathcal{V}_8^{\leftarrow}$-cascas deste conjunto encontraremos ao todo 38 cascas (os pontos pretos da Figura 3.5). Se neste mesmo conjunto, tomássemos as $\mathcal{V}_{(h)}^{\leftarrow}$-cascas no sentido horizontal, encontraríamos ao todo 26 cascas. Já, se tomássemos as $\mathcal{V}_{(v)}^{\leftarrow}$-cascas no sentido vertical, teríamos 29 cascas. Note que um conjunto de cascas qualquer é relativamente compacto.

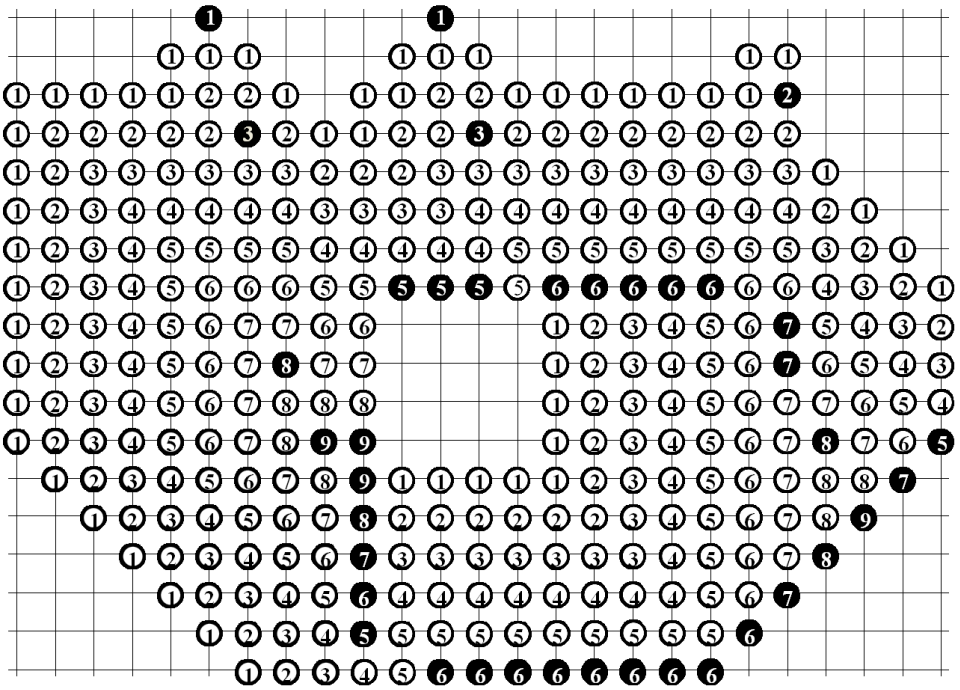


Figura 3.5: Os pontos pretos são as $\mathcal{V}_8^{\leftarrow}$-cascas da Fig. 3.4.

Denote por $\mathcal{Z}^V(X)$ o conjunto de todas as **V**-cascas de X. Observe que, basicamente, o conjunto $\mathcal{Z}^V(X)$ corresponde ao esqueleto da imagem X, segundo o conjunto de adjacência V.

Com a programação dinâmica, podemos implementar um algoritmo **Shell**(X, V) para encontrar o conjunto $\mathcal{Z}^V(X)$.

Algoritmo 3: Shell(X, V) {

1: $\mathcal{Z}^V(X) \leftarrow \emptyset$;
2: Execute **setDensity**(X, V);
3: Para todo $x \in X$ faça
4: Se $(\forall v \in V^t : \Delta_X^V(x+v) \leq \Delta_X^V(x))$ então
5: $\mathcal{Z}^V(X) \leftarrow \mathcal{Z}^V(X) \cup \{x\}$;
6: Devolva $\mathcal{Z}^V(X)$;

}

3.3.1 Análise de Complexidade

Como podemos observar, é mais conveniente armazenar o conjunto $\mathcal{Z}^V(X)$ em uma lista ligada. Para a análise o consumo de tempo do algoritmo Shell, considere a seguinte tabela:

Linha do Algoritmo	Consumo de Tempo
1	$\Theta(1)$
2	$\Theta(\|X\| \cdot \|V\|)$
3	$\Theta(\|X\|)$
4	$\Theta(\|X\| \cdot \|V\|)$
5	$O(\|X\| \cdot \|V\|)$
6	$\Theta(1)$
Total	$\Theta(\|X\| \cdot \|V\|)$

Todos os pontos de X são visitados anteriormente pelo algoritmo setDensity, cuja complexidade é $\Theta(|X| \cdot |V|)$. Assim, a complexidade computacional do algoritmo Shell é $\Theta(|X| \cdot |V|)$. Se considerarmos que $|V|$ é um valor constante, então o algoritmo Shell consome tempo linear.

Podemos ainda, construir o algoritmo **setDensityAndShell**(X, V) que ao mesmo tempo calcula a Transformada da Densidade e devolve o conjunto $\mathcal{Z}^V(X)$ usando uma *fila* e uma *memoized table*.

Algoritmo 4: setDensityAndShell(X, V) {

```
    % Considere que para todo p ∉ X, tem-se que Δ_X^V(p) = 0
1:  cont^V(X) ← Contour(X, V);
2:  Para todo x ∈ X faça
3:      Se (x ∈ cont^V(X)) então
4:          Δ_X^V(x) ← 1;
        Senão
5:          Δ_X^V(x) ← +∞;
6:  Z^V(X) ← X;
7:  fila ← cont^V(X);
8:  Enquanto fila ≠ ∅ faça
9:      p ← desenfilera(fila);
10:     Para todo v ∈ V^t faça
11:         Se (Δ_X^V(p) + 1 < Δ_X^V(p + v)) então
12:             Δ_X^V(p + v) ← Δ_X^V(p) + 1;
13:             enfilera(p + v, fila);
14:             Z^V(X) ← Z^V(X) \ {p};
15:  Devolva Z^V(X);
```

}

Vale salientar que, uma vez que um ponto p' entra na fila, este possui V-densidade minimal, e não entra novamente na fila em iterações posteriores.

Sendo assim, segue a tabela de consumo de tempo para este algoritmo:

Linha do Algoritmo	**Consumo de Tempo**
1	$\Theta(\|X\| \cdot \|V\|)$
2–5	$\Theta(\|X\|)$
6	$\Theta(\|X\|)$
7	$\Theta(1)$
8–9	$\Theta(\|X\|)$
10–11	$\Theta(\|X\| \cdot \|V\|)$
12–14	$O(\|X\| \cdot \|V\|)$
15	$\Theta(1)$
Total	$\Theta(\|X\| \cdot \|V\|)$

Como podemos observar, este algoritmo é linear, considerando que $|V|$ é constante.

Com todas estas definições, podemos demarcar algumas proposições utilizadas pelo algoritmos de erosão e dilatação que apresentaremos no próximo capítulo.

Propriedade 3.3.2 *Seja $X \subseteq \mathbb{Z}^n$. Valem as seguintes propriedades:*

1. $\mathcal{Z}^V(X) \subseteq X$.
2. $COB_X^V(p) \subseteq X$, *para qualquer* $p \in \mathbb{Z}^n$.
3. $X = \bigcup_{\zeta \in \mathcal{Z}^V(X)} COB_X^V(\zeta)$.

Proposição 3.3.3 *Sejam $X, Y, V \subseteq \mathbb{Z}^n$ e seja $h \in X$. Então $\Delta_X^V(h) \leq \Delta_Y^V(h)$ se, e somente se, h em X é dominado por h em Y, ou seja, $COB_X^V(h) \subseteq COB_Y^V(h)$.*

Prova:*($\Rightarrow$) Para qualquer $q \in COB_X^V(h)$ vale que $d^V(h,q) < \Delta_X^V(h)$. Como $\Delta_X^V(h) \leq \Delta_Y^V(h)$, para todo $q \in COB_X^V(h)$ temos que $d^V(h,q) < \Delta_Y^V(h)$. Logo $COB_X^V(h) \subseteq COB_Y^V(h)$.*
($\Leftarrow$) Considere os seguintes casos:

(a) $COB_X^V(h) = COB_Y^V(h)$. *Neste caso é evidente que $\Delta_X^V(h) = \Delta_Y^V(h)$.*

(b) $COB_X^V(h) \subset COB_Y^V(h)$. *Seja $q \in COB_Y^V(h)$ tal que $q \notin COB_X^V(h)$. Então $\Delta_X^V(h) \leq d^V(h,q)$, caso contrário $q \in COB_X^V(h)$, uma contradição. Então $\Delta_X^V(h) \leq d^V(h,q) < \Delta_Y^V(h)$ e $\Delta_X^V(h) < \Delta_Y^V(h)$.*

Logo, se $COB_X^V(h) \subseteq COB_Y^V(h)$ então $\Delta_X^V(h) \leq \Delta_Y^V(h)$. $\diamond$

Proposição 3.3.4 *Todo ponto p de X é coberto por alguma casca de $\mathcal{Z}^V(X)$.*

Prova:*Ou p é coberto por alguma casca de $\mathcal{Z}^V(X)$ ou não é coberto por ninguém diferente de p, e por definição, p é a um casca de $\mathcal{Z}^V(X)$ sendo coberta por ela mesma.* $\diamond$

Propriedade 3.3.5 *Sejam $X, V \subseteq \mathbb{Z}^n$ e $h \in X$.*

Então $COB_X^V(h) = (mV)_h$, onde $m = \Delta_X^V(h) - 1$.

Prova: *Relembrando: $COB_X^V(h) = \{x \in X : d^V(h,x) < \Delta_X^V(h)\}$. Note que para todo $v \in V$ tal que $p = h + v$ (p é V-adjacente a h) temos que $\Delta_X^V(p) \geq \Delta_X^V(h) - 1$, senão há uma contradição na própria definição de $\mathbf{V}$-densidade. Sejam $m = \Delta_X^V(h) - 1$ e $Y_p = ((m-1)V)_p$ para cada p $\mathbf{V}$-adjacente a h. Segue das aplicações diretas das definições de $\mathbf{V}$-densidade e $\mathbf{V}$-casca que $\Delta_{Y_p}^V(p) = m$ e $\mathcal{Z}^V(Y_p) = \{p\}$. Assim, temos que $\Delta_{Y_p}^V(p) = \Delta_X^V(h) - 1 \leq \Delta_X^V(p)$, e como Y_p possui somente uma V-casca, $Y_p \subseteq COB_X^V(p)$ (ver Prop. 3.3.3). Sabe-se que $d^V(h,p) = 1, \forall p$ V-adjacente a h. Além disso,*

$\forall q \in Y_p$ *temos que* $d^V(p,q) < m$*(ver definição de cobertura). Logo, pela desigualdade triangular temos que* $d^V(h,q) \leq d^V(h,p) + d^V(p,q) < m+1 = \Delta_X^V(h)$*, ou seja,* $COB_X^V(h)$ *cobre todos os pontos de* Y_p*, para todos os pontos* p V*-adjacentes a* h*. Assim,* $COB_X^V(h) = \{q \in Y_p : \forall v \in V$ *temos que* $p = h + v\} = \bigcup_{v \in V}(Y_h)_v = Y_h \oplus V = ((m-1)V)_h \oplus V = (mV)_h$. $\diamond$

A Figura 3.6 mostra um exemplo ilustrativo da Propriedade 3.3.5. Note que o ponto preto x desta imagem possui **V**-densidade $\Delta_X^V(x) = 3$, ou seja, $(2V)_h = COB_X^V(h) \subseteq X$.

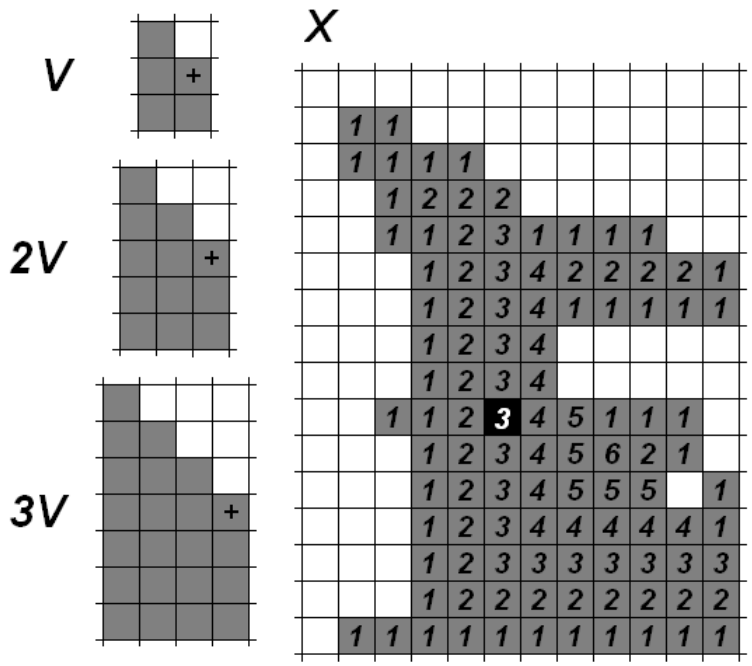


Figura 3.6: À esquerda temos o conjunto de adjacência V, $2V$ e $3V$. À direita, observe que na imagem X, o ponto preto é origem de um $2V$ por possui **V**-densidade igual a 3.

Note que o conjunto de cascas $\mathcal{Z}^V(X)$ do conjunto X (ou imagem binária), pode ser visto, sem perda de generalidade, como uma representação compacta deste conjunto.

Capítulo 4

Algoritmos para Erosão e Dilatação

Este capítulo se dedica à revisão de algoritmos rápidos de dilatação e erosão e a implementação de novos algoritmos embasados em conceitos definidos no capítulo anterior.

4.1 Revisão de Algoritmos para Erosão e Dilatação

Segundo a abordagem *clássica*, executar a erosão binária de uma imagem binária $X \in \mathcal{P}(\mathbb{Z}^n)$ por um elemento estruturante $B \in \mathcal{P}(\mathbb{Z}^n)$ equivale a realizar uma seqüência de interseções da imagem de entrada X transladada pelos pontos de B^t. O algoritmo basicamente realiza a seguinte operação:

$$\varepsilon_B(X) = \bigcap_{h \in B^t} X_h.$$

Similarmente, executar a dilatação binária de X por B equivale a realizar uma seqüência de uniões da imagem de entrada X transladada pelos pontos de B. O algoritmo basicamente realiza a seguinte operação:

$$\delta_B(X) = \bigcup_{h \in B} X_h.$$

Porém, no pior caso, tais algoritmos clássicos calculam a erosão/dilatação em tempo $O(|X| \cdot |B|)$. Na prática, tal complexidade é ruim, já que na prática as imagens de entrada contém muitos pontos.

Uma outra abordagem, proposta por Xu [47], consiste em decompor elementos estruturantes convexos em uma seqüência de erosões/dilatações por elementos estruturantes primitivos (com janelas

3×3) centrados na origem. Tais propriedades advêm da adição de Minkowski:

$$X \oplus B = X \oplus (B_1 \oplus B_2 \oplus \cdots \oplus B_N) = ((((X \oplus B_1) \oplus B_2) \oplus \cdots) \oplus B_N)$$

e da subtração de Minkowski:

$$X \ominus B = X \ominus (B_1 \oplus B_2 \oplus \cdots \oplus B_N) = ((((X \ominus B_1) \ominus B_2) \ominus \cdots) \ominus B_N),$$

onde $B = B_1 \oplus B_2 \oplus \cdots \oplus B_N$.

Tal propriedade é propagada à erosão e à dilatação, conforme visto na propriedade de decomposição de elementos estruturante (Prop. 2.3.1) mostrada na Seção 2.3.

A decomposição do elemento estruturante é um importante fator de redução na complexidade computacional destes algoritmos. No entanto, existem elementos estruturantes que não podem ser decompostos, levando a complexidade no pior caso a $O(|X| \cdot |B|)$. Mas de uma forma geral, para elementos estruturantes convexos conseguimos obter resultados bastante relevantes. Por exemplo, um diamante $N \times N$ pode ser decomposto por α cruzes 3×3, assim como um quadrado $N \times N$ pode ser decomposto por α quadrados 3×3, onde $N = 2\alpha + 1$. Nestes casos, a complexidade no pior caso passa a ser $O(|X| \cdot \alpha)$, uma vez que cada operação morfológica cujos elementos estruturantes têm tamanho fixo (3×3) se realiza em tempo constante. Assintoticamente, $\alpha = O(N)$, e como $|B| = N \cdot N$ temos que $\alpha = O(|B|^{\frac{1}{2}})$. Por isso, costuma-se dizer que estes algoritmos de decomposição consomem tempo proporcional a $O(|X| \cdot |B|^{\frac{1}{2}})$ no pior caso.

Sabe-se ainda que em algoritmos de decomposição logarítmica [42], conseguimos algoritmos cuja complexidade computacional chega a $O(|X| \cdot \log(|B|^{\frac{1}{2}}))$.

Zhuang e Haralick [49] apresentaram um algoritmo que usa árvore de buscas para a decomposição de um elemento estruturante arbitrário. Existem vários outros métodos e técnicas de decomposição, como por exemplo, a decomposição linear de elementos estruturantes convexos [18], uma técnica de otimização combinatória para decomposições [19] e outros métodos, tais como a decomposição baseada em programação linear inteira [48] e baseada em algoritmos genéticos [37].

Todavia, podemos representar estas operações de outras maneiras. Uma vez que a dilatação e a erosão são tidas como mapeamentos 'janelados' *(W-operadores)* entre reticulados completos, podemos representá-los através de funções Booleanas. Uma representação de uma função Booleana por uma estrutura baseada em decisão foi introduzida por Lee [23] e depois foi popularizada por Akers [2] sob o nome de Diagrama de Decisão Binária, ou *BDD* (do inglês, *Binary Decision Diagram*). Algoritmos para manipulação de BDD são descritos em [15]. Implementações eficientes de BDDs podem ser encontradas em [13]. Recentemente, foram desenvolvidas aplicações de BDDs em Processamento Digital de Imagens [8, 34].

Resumidamente, o BDD de uma função Booleana que representa um W-operador pode ser visto como um grafo dirigido acíclico enraizado com dois tipos de nós: internos e terminais. Cada nó

terminal v tem como um atributo um valor $value(v) \in \{0, 1\}$. Os nós internos v têm dois nós filhos, $low(v)$ e $high(v)$. Cada nó interno v é rotulado com um índice de uma das variáveis de entrada $index(v) \in \{1, 2, \ldots, |W|\}$ da janela W. No caso de imagens binárias no espaço $\mathbb{Z}^n$, o índice é exatamente uma coordenada cartesiana deste espaço.

Para uma certa atribuição de valores 0 ou 1 aos nós do BDD (avaliando-se a janela do elemento estruturante sobre a imagem), o valor da função é determinado pelo caminho no grafo da raiz até um nó terminal: para cada nó interno v com $index(v) = i$, se $x_i = 0$, então utiliza-se o arco para o nó $low(v)$. Caso contrário (i.e., $x_i = 1$), o arco para o nó $high(v)$ é utilizado. O valor da função é dado pelo valor do nó terminal, onde o caminho termina.

Sendo assim, podemos a partir das definições alternativas de erosão e dilatação por um elemento estruturante B, construir as BDDs $\mathbf{G}_{\varepsilon_B}$ e $\mathbf{G}_{\delta_B}$, respectivamente, conforme o exemplo mostrado na Figura 4.1. Além disso, pode-se compor grafos para quaisquer W-operadores, uma vez definidas operações morfológicas sobre as BDDs [25, 26, 27].

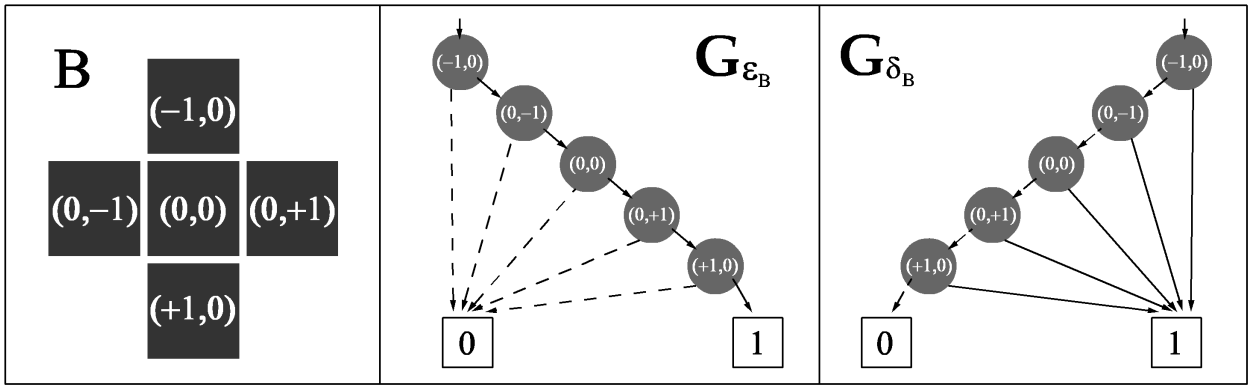


Figura 4.1: As imagens acima ilustram $\mathbf{G}_{\varepsilon_B}$ e $\mathbf{G}_{\delta_B}$, respectivamente.

Note que um BDD implementa basicamente as definições equivalentes de dilatação e erosão como seguem:

$$\varepsilon_B(X) = \{h \in \mathbb{Z}^n : B_h \subseteq X\} \text{ e } \delta_B(X) = \{h \in \mathbb{Z}^n : {B^t}_h \cap X \neq \emptyset\}.$$

É evidente que o BDD é uma técnica mais adequada quando queremos compor tais W-operadores, ao invés de utilizá-lo somente para implementar dilatação e erosão. Contudo, ainda que as implementações de dilatação e erosão que usam BDDs sejam bem rápidos na prática, no pior caso elas são $O(|X| \cdot |B|)$.

Um outro método para calcular erosão e dilatação, proposto por I. Young [21], consiste em usar *Run-Length Encoding*, ou RLE como é conhecida (semelhante ao método de codificação de

intervalos [31]). O RLE é uma codificação que comprime uma seqüência de valores iguais de uma cadeia de valores quaisquer. Tal codificação pode ser utilizada com bastante propriedade sobre as imagens binárias. Por exemplo, considere uma seqüência de valores de uma linha de uma imagem qualquer conforme mostrada abaixo. Cada ∘ é um pixel de fundo e cada • é um pixel do objeto de uma imagem qualquer. Note que existem grandes "runs" (seqüência contínua) de pixels do *background* e poucas "runs" de pixels do *foreground*:

∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ • ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ • • • ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘

Se aplicarmos o algoritmo RLE de compressão de dados sobre esta cadeia de valores hipotética, obteremos o seguinte resultado:

$$10 \circ \bullet 12 \circ 3 \bullet 12 \circ$$

A compressão é evidente. Kim [46] propôs um algoritmo para calcular a erosão e a dilatação utilizando apenas as "runs" do *foreground* tanto do RLE da imagem de entrada, quanto do RLE do elemento estruturante.

Sendo assim, a erosão e a dilatação passam a ter complexidade no pior caso $O(|RLE(X)| \cdot |RLE(B)|)$, onde $RLE(Y)$ corresponde ao número de "runs" resultante da codificação RLE de uma imagem Y.

Há também uma abordagem que utiliza contornos das imagens para calcular a dilatação. Zamperoni [32] foi um importante precursor desta abordagem, e implementou uma técnica para calcular erosão e dilatação usando o código da cadeia do contorno das imagens para um elemento estruturante convexo. Mas Vincent [44] estendeu o conceito para elementos estruturantes arbitrários. Segundo ele, calcular uma dilatação de X por um elemento estruturante B equivale (basicamente) a calcular a dilatação do contorno de X por B. Ele demonstrou a seguinte propriedade: *Sejam* $X, B \subseteq \mathbb{Z}^n$. *Seja* $cont(X)$ *o contorno (4-)8-conexo de* X. *Considere que* B *é composto por* $B_1, B_2, \ldots, B_m$ *componentes (4-)8-conexas. Para cada* $i \in [1..m]$, *seja* $p_i \in B_i$. *Sendo assim:*

$$\delta_B(X) = \bigcup_{i \in [1..m]} X_{p_i} \cup \delta_B(cont(X)).$$

Nota-se que realizar a operação $\delta_B(X)$ se restringe a realizar a operação $\delta_B(cont(X))$ com algumas uniões. Como a dilatação é comutativa, podemos basicamente reduzir a operação $\delta_B(X)$ à operação $\delta_{cont(B)}(cont(X))$.

O processo de extração de contorno dos conjuntos consome tempo linear. Sendo assim, o pior caso deste algoritmo é $O(|cont(X)| \cdot |cont(B)|)$, além do tempo de pré-processamento dos contornos que é $O(|X| + |B|)$.

Existem ainda algoritmos muito rápidos na prática, que usam o paralelismo aparente da CPU ao executar operações de E e OU lógicas [11, 14]. Estes algoritmos efetuam as operações de erosão e

dilatação do mesmo modo que os algoritmos Clássicos, porém, aproveitando o paralelismo existente nas operações lógicas (E/OU) executadas pela CPU entre números inteiros. Cada número inteiro é considerado um vetor de bits (*bit-vector*). O tempo de tais operações lógicas é constante.

Considerando que um número inteiro na CPU tem η bits (normalmente 32 bits), podemos representar uma seqüência de η pixels de imagem de entrada f (incluindo pixels de fundo e do objeto da imagem), por apenas um único inteiro.

O custo de se realizar uma operação de interseção/união entre duas imagens de tamanho N passa a ser N/η em vez de N. Logo, o custo computacional para efetuarmos operações de erosão e dilatação de um conjunto X pelo elemento estruturante B é $O(|B| \cdot |X|/\eta)$.

A partir desta revisão, vamos propor dois algoritmos para erosão que utilizam o pré-processamento de densidades, utilizando pressupostos de técnicas de busca de padrão em textos; e um algoritmo para dilatação usando o conjunto de cascas das imagens de entrada, a saber:

- *Jump-miss Erosion*: Erosão com saltos por erros;
- *Jump-hit Erosion*: Erosão com saltos por acertos;
- *Shell Dilation*: Dilatação por cascas.

4.2 Novos Algoritmos para Erosão e Dilatação

Nesta seção apresentaremos dois tipos de algoritmos: algoritmos para erosão usando pressupostos de busca de padrão em textos e um algoritmo de dilatação que utiliza apenas os conjuntos de cascas da imagem de entrada e do elemento estruturante.

Um grande entrave na implementação de algoritmos rápidos aplicados aos conjuntos é saber se um elemento $x \in \mathbb{Z}^n$ pertence ou não a um conjunto X. Outro problema é definir uma ordenação sobre os pontos de conjunto. A escolha de uma estrutura de dados eficiente é fundamental para a resolução destes problemas. Podemos representar um conjunto de diversas formas *(listas ordenadas, árvores, matrizes, vetores de bits,...)*. Entretanto, como grande parte das aplicações em morfologia matemática giram em torno de imagens binárias, opta-se, preferivelmente, representar os conjuntos por uma *matriz multidimensional* D_X correspondente à função indicadora de subconjuntos $1_X(x)$, para algum $x \in \mathbb{Z}^n$. Neste tipo de estrutura de dados, em tempo constante sabe-se se x pertence ou não a X. Em geral, a matriz multidimensional D_X possui dimensões equivalentes as do menor retângulo que envolve (*Bounding Box*) o objeto X.

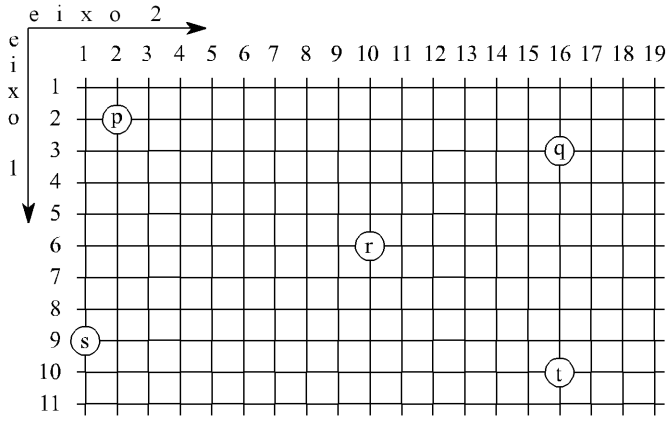
Ao longo deste capítulo, propomos uma relação de ordem lexicográfica $\leqslant^L : \mathbb{Z}^n \times \mathbb{Z}^n$ definida sobre os pontos de $\mathbb{Z}^n$. Tal ordenação é feita sobre as coordenadas de cada ponto deste conjunto, segundo uma *ordem de prioridades* sobre as coordenadas do espaço $\mathbb{Z}^n$. Tal ordem de prioridades é

representada por uma permutação $L = (k_1, k_2, \ldots, k_n)$ de 1 a n. A partir desta ordem, estabelece-se que um ponto $p \in \mathbb{Z}^n$ é menor que um ponto $q \in \mathbb{Z}^n$ ($p <^L q$) se, e somente se, existe um inteiro $i \in [1..n]$, tal que $p_{k_i} < q_{k_i}$ e para todo $j < i$ temos que $p_{k_j} = q_{k_j}$. Neste caso, dizemos também que q é maior que p ($q >^L p$). Caso $x \not<^L y$, dizemos que $x \geq^L y$, bem como se $x \not>^L y$, dizemos que $x \leq^L y$.

Com isto, podemos ordenar um conjunto crescente ou decrescentemente segundo as prioridades definidas por $L = (k_1, k_2, \ldots, k_n)$. Portanto, um conjunto X se encontra em ordem lexicográfica *crescente* $(X, <^L)$ ou *decrescente* $(X, >^L)$ se, e somente se, todo ponto $x \in X$ que ocorre primeiro que $y \in X$ em tal ordem, tem-se que $x <^L y$ ou $x >^L y$, respectivamente.

Note que as matrizes multidimensionais permitem acesso aos elementos da imagem em ordem. A prioridade das coordenadas é determinada pela composição dos laços aninhados que percorrem tal estrutura de dados.

A Figura 4.2 nos mostra um exemplo de ordenação entre pontos no espaço $\mathbb{Z}^2$. Conforme indicado, o eixo 1 corresponde ao eixo vertical e o eixo 2 corresponde ao eixo horizontal. A lista de prioridades $L_1 = (1, 2)$ ordena os pontos prioritariamente no eixo vertical, enquanto que a lista $L_2 = (2, 1)$, ordena no eixo horizontal. Visto que $p = (2, 2)$, $q = (3, 16)$, $r = (6, 10)$, $s = (9, 1)$ e $t = (10, 16)$, temos que $p <^{L_1} q <^{L_1} r <^{L_1} s <^{L_1} t$ assim como $s <^{L_2} p <^{L_2} r <^{L_2} q <^{L_2} t$.



$L_1 = (1, 2)$

$p <^{L_1} q <^{L_1} r <^{L_1} s <^{L_1} t$

$L_2 = (2, 1)$

$s <^{L_2} p <^{L_2} r <^{L_2} q <^{L_2} t$

Figura 4.2: Os pontos $p, q, r, s, t \in \mathbb{Z}^2$ ordenados lexicograficamente segundos as listas de prioridade L_1 e L_2.

Vamos então, apresentar os algoritmos de erosão que utilizam técnicas de *string-matching* [40].

4.2.1 Erosão com pressupostos de *String-Matching*

Nesta seção, apresentaremos dois algoritmos que calculam a erosão rapidamente, utilizando a Transformada da Densidade e as Cascas do elemento estruturante [1] combinadas com técnicas de *String-Matching*.

Para se ter uma intuição de como funciona estes algoritmos de erosão, é necessário entender um pouco do funcionamento de alguns algoritmos de busca de padrões (algoritmos *String-Matching*) básicos.

Se tratando em busca de padrões em texto, considere que Σ é um *alfabeto* (conjunto finito). Tanto o *padrão* quanto o *texto* de entrada são compostos por concatenação de elementos de Σ. Um algoritmo de busca de padrões basicamente tenta encontrar posições na qual o padrão está contido no texto. Um algoritmo ingênuo para encontrar padrões P em um texto T gasta tempo proporcional a $\Theta(|P| \cdot |T|)$ para efetuar a busca. Entretanto, existem algoritmos que otimizam esta busca, utilizando técnicas de *pré-processamento*.

Tais pré-processamentos buscam codificar as entradas de modo que o espaço de busca se torne mais reduzido. Alguns algoritmos executam um *pré-processamento* sobre o texto de entrada (árvore de sufixos [45] em tempo linear [41]), outros aplicam sobre o padrão (KMP [16]), ou sobre ambos (busca de expressões regulares [5]). Em geral, os pré-processamentos consomem tempo linear no tamanho da entrada.

Outra característica interessante presente em alguns algoritmos de busca de padrão em textos é a propriedade de realizar saltos na busca, a medida que se garante que a entrada do texto não será mais encontrada no padrão (algoritmo de Boyer-Moore [33]). Tais saltos são conhecidos como *jumps*.

Com base nestas informações, o objetivo desta seção é reduzir o problema de busca de padrões textuais ao problema de realizar a erosão morfológica binária. Neste caso, o alfabeto $\Sigma = \{0, 1\}$ corresponde aos pixels de uma imagem f qualquer (ver Seção 2), o padrão corresponde ao elemento estruturante e o texto corresponde a uma imagem de entrada.

Sejam $X, B \subseteq \mathbb{Z}^n$. Observe novamente a definição alternativa da erosão morfológica binária: $\varepsilon_B(X) = \{h \in \mathbb{Z}^n : B_h \subseteq X\}$. Ou seja, se todo ponto de B_h pertence a X, então $h \in \varepsilon_B(X)$.

O cálculo da erosão é reduzido a um problema de busca dos pontos do elemento estruturante B dentro da imagem de entrada X.

Assim como na busca textual, a busca deve ocorrer em ordem dentro da matriz que armazena

[1] Relembrando: tanto a Transformada da Densidade, quanto encontrar o conjunto maximal de cascas consomem tempo (quase) linear.

a imagem X (isto é D_X). No caso de textos, os algoritmos *string-matching* efetuam as buscas do padrão sobre as linhas do texto em um sentido específico dentro das cadeias de caracteres (da esquerda para direita ou vice-versa). Observe por sua vez, que a ordem em que as linhas serão avaliadas é irrelevante, visto que todas as linhas são processadas independentemente pelo algoritmo.

Nas imagens binárias as buscas também ocorrem sobre "linhas" (seqüencia de pixels). Entretanto, neste caso a ordem da busca pode ser realizada sobre "linhas" orientadas em qualquer *direção* e *sentido*.

A direção e o sentido da busca podem ser representados por um vetor $k \in \mathbb{Z}^n$.

Neste trabalho, por simplificação, considere que a direção e o sentido do percurso (sentido da busca) serão representados pelo vetor k de um conjunto $\mathcal{V}_{(1,e)} \subseteq \mathbb{Z}^n$. Neste caso, o vetor k pertence a algum eixo $e \in [1..n]$ de $\mathbb{Z}^n$. Tal vetor k é dependente do eixo e no qual temos que $k_e \neq 0$ (relembrando que $k_e \in (-1, 1)$). Assim, cada componente i de k tem valor

$$|k_i| = \begin{cases} 1, & \text{se } i = e \\ 0, & \text{caso contrário.} \end{cases}$$

A relação entre este eixo e e a ordem lexicográfica sobre um conjunto qualquer é dada por uma lista de prioridades L definida como:

$$L = (1, 2, \ldots, e-2, e-1, n, e+1, e+2, \ldots, n-2, n-1, e),$$

onde $e \in [1..n]$ é o eixo tal que $|k_e| = 1$. Vale ressaltar que se $e = 1$ então $L = (n, 2, 3, \ldots, n-2, n-1, 1)$.

O sentido do percurso é definido pelo sinal de k_e. Dizemos que o sentido do percurso é *direto* quando $k_e = 1$; e é *contrário* quando $k_e = -1$.

No espaço $\mathbb{Z}^2$, o vetor $k = (0, 1)$, considerando que e é o eixo horizontal (eixo 2) indica que a busca ocorrerá em *raster order*, cuja lista de prioridades $L = (1, 2)$, enquanto que o vetor $k = (0, -1)$ indica que a busca ocorrerá em *anti-raster order*.

O conjunto de adjacência V utilizado nos passos de pré-processamento das imagens de entrada (Transf. Densidade, Conjunto de Cascas...) destes algoritmos de erosão é definido como o conjunto $\mathcal{V}^{\leftarrow}_{(1,e)} \subseteq \mathbb{Z}^n$ no qual $k \in \mathcal{V}_{(1,e)}$. Em geral, o vetor k é escolhido de forma a obter o menor número possível de cascas em $\mathcal{Z}^{\mathcal{V}^{\leftarrow}_{(1,e)}}(B)$.

Conforme levantado, para busca, os algoritmos de erosão propostos por este trabalho realiza um pré-processamento sobre as imagens de entrada. Tal pré-processamento é dividido em duas fases:

1. Encontrar as $\mathbf{V}$-densidades dos pontos da imagem de entrada X.

2. Encontrar as **V**-cascas do elemento estruturante B.

Observe na Figura 4.3 uma representação gráfica do sentido dos percursos sobre um conjunto devidamente pré-processado pela Transformada da Densidade, utilizando como conjunto de adjacência o conjunto $V = \{(0, -1)\}$. O sentido do crescimento da V-densidade desta figura ocorre da esquerda para direita. Os vetores k_1 e k_2 correspondem dois sentidos de busca distintos. Neste caso, o sentido indicado por k_1 é o mesmo sentido do crescimento da V-densidade, enquanto que o sentido indicado por k_1 corresponde ao sentido contrário ao crescimento da V-densidade. Por este modo, dizemos que uma busca que ocorre no sentido k_1 é uma busca direta (percurso direto) e uma busca orientada no sentido k_2 é uma busca inversa (percurso contrário).

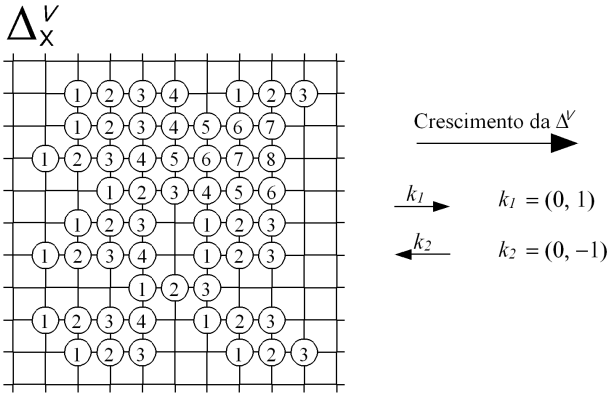


Figura 4.3: Uma imagem submetida a Transformada da Densidade para o conjunto de adjacência V. Note que o aumento das V-densidades ocorrem no mesmo sentido da busca indicada pelo vetor k_1, e ocorrem no sentido contrário ao da busca indicada pelo vetor k_2.

Este trabalho propõe dois algoritmos de erosão, cada um dedicado a tipo específico de busca: na *erosão direta*, também conhecido como *jump-miss erosion*, o mesmo sentido da busca é direto e na *erosão inversa*, também conhecido como *jump-hit erosion*, sentido da busca é contrário.

Vamos então à apresentação do algoritmo de erosão *jump-miss erosion*.

Jump-miss Erosion: a Erosão Direta

Conforme dito, o algoritmo de erosão direta realiza buscas do elemento estruturante sobre a imagem de entrada no sentido direto, de acordo com um vetor $k \in \mathcal{V}_{(1,e)}$ escolhido. Ademais, o conjunto de vizinhança V é dado por $\mathcal{V}^{\leftarrow}_{(1,e)} \subseteq \mathbb{Z}^n$, segundo o eixo $e \in [1..n]$, logo $k \in \mathcal{V}^{\leftarrow}_{(1,e)}$.

Entretanto, considere algumas proposições importantes para o desenvolvimento do algoritmo.

Proposição 4.2.1 *Sejam $X, B \subseteq \mathbb{Z}^n$ e V um conjunto de adjacência qualquer. Então $B \subseteq X$ se, e só se, para cada* **V***-casca ζ de $\mathcal{Z}^V(B)$ existe um ponto x em X tal que $COB_B^V(\zeta) \subseteq COB_X^V(x)$, ou seja, x em X* **V***-domina o ponto ζ em B.*

Prova: *($\Rightarrow$) Se $B \subseteq X$, então toda* **V***-casca ζ de $\mathcal{Z}^V(B)$ e seus respectivos pontos* **V***-cobertos em B pertencem a X. Sendo assim, o ponto $x = \zeta$ é um ponto em X que* **V***-domina o ponto ζ em B. ($\Leftarrow$) Pela Proposição 3.3.4, todo ponto de B é* **V***-coberto em B por alguma $\zeta \in \mathcal{Z}^V(B)$. Como para cada* **V***-casca ζ de $\mathcal{Z}^V(B)$ existe um ponto x em X tal que $COB_B^V(\zeta) \subseteq COB_X^V(x)$, temos finalmente que todo ponto de B também pertence a X.*

De acordo com tal proposição, dado um $h \in \mathbb{Z}^n$ de modo que cada **V**-casca $\zeta \in \mathcal{Z}^V(B_h)$ em B_h é **V**-dominada por algum ponto x em X, então $h \in \varepsilon_B(X)$.

Proposição 4.2.2 *Sejam $B, V \subseteq \mathbb{Z}^n$ e $h \in \mathbb{Z}^n$. Então existe uma bijeção entre as* **V***-cascas ζ de $\mathcal{Z}^V(B)$ e as* **V***-cascas ζ' de $\mathcal{Z}^V(B_h)$, de modo que $\zeta + h = \zeta'$ e $\Delta_B^V(\zeta) = \Delta_{B_h}^V(\zeta')$.*

Prova: *Claramente, ao transladar o conjunto B pelo vetor h, estamos também transladando as* **V***-cascas ζ de $\mathcal{Z}^V(B)$. De fato, uma vez que o conjunto $\mathcal{Z}^V(B_h) = \{\zeta + h : \zeta \in \mathcal{Z}^V(B)\}$ é o conjunto de todas as* **V***-cascas de B_h. Ademais, transformada da densidade é i.t., o que garante que $\Delta_B^V(\zeta) = \Delta_{B_h}^V(\zeta')$.*

Portanto, não é necessário calcular a Transformada da Densidade para o conjunto B_h, para cada h distinto. Ou seja, o pré-processamento da Transformada da Densidade e de Cascas é único, tanto para o elemento estruturante, quanto para a imagem de entrada.

Propriedade 4.2.3 *Segundo a Proposição 3.3.4, todos os pontos de B são* **V***-cobertos pelas suas respectivas* **V***-cascas $\zeta \in \mathcal{Z}^V(B)$. Além disso, a Proposição 3.3.3 nos garante que $\Delta_B^V(\zeta) \leq \Delta_X^V(\zeta)$ se, e somente se, $COB_B^V(\zeta) \subseteq COB_X^V(\zeta)$.* ***Logo, se para algum $h \in \mathbb{Z}^n$, para cada* V***-casca ζ de $\mathcal{Z}^V(B_h)$ vale que $\Delta_{B_h}^V(\zeta) \leq \Delta_X^V(\zeta)$, então $B_h \subseteq X$ e, por conseguinte, $h \in \varepsilon_B(X)$.***

Tal conclusão nos leva a entender que não é necessário verificar se todos os pontos de B_h estão contidos em X: *basta examinar as cascas de $\mathcal{Z}^V(B_h)$.*

Por exemplo, suponha que queremos construir um algoritmo cujo elemento estruturante B é um único quadrado. Suponha ainda que o conjunto de adjacência é o conjunto $\mathcal{V}_8^{\leftarrow} \subseteq \mathbb{Z}^n$. Logo, este contém só uma casca 8-conexa anterior ζ (a extremidade inferior direita do quadrado) de densidade $\Delta_B^{\mathcal{V}_8^{\leftarrow}}(\zeta)$. Sendo assim, a erosão de uma imagem X por este elemento estruturante B consiste em encontrar pontos $x \in X$, tais que $\Delta_B^{\mathcal{V}_8^{\leftarrow}}(\zeta) \leq \Delta_X^{\mathcal{V}_8^{\leftarrow}}(x + \zeta)$. Com isto, a erosão de uma imagem qualquer por um quadrado de tamanho qualquer se realiza em tempo linear no tamanho da entrada.

Observe ainda, que a busca das cascas de $\mathcal{Z}^V(B_h)$ em X não precisa percorrer todos os pontos h da matriz que armazena o conjunto X, isto é, D_X. Se a origem está em B, então $\varepsilon_B(X) \subseteq X$, ou seja, se B contém a origem, basta percorrer somente pelos pontos $h \in X$. Ademais, se a origem não estiver em B, ao fixarmos um ponto $b \in B$, então $\varepsilon_{B_{-b}}(X) = (\varepsilon_B(X))_b$. Logo $(\varepsilon_B(X))_b \subseteq X$, uma vez que $o \in \varepsilon_{B_{-b}}(X)$. Neste caso, calcula-se $((\varepsilon_B(X))_b)_{-b} = \varepsilon_B(X)$ (ver Propriedade 2.3.1). Desta forma, por simplificação, considere que nos algoritmos de erosão, a origem o está em B.

Só em buscar as **V**-cascas de um elemento estruturante B em um conjunto X já obtemos um algoritmo de erosão bastante rápido. Poderíamos, por exemplo, buscar pontos de $\mathcal{Z}^V(B)$ em pontos $x \in X$ para os quais $\Delta_B^V(\zeta_{\max}) \leq \Delta_X^V(x + \zeta_{\max})$, onde $\zeta_{\max} \in \mathcal{Z}^V(B)$ é uma **V**-casca de **V**-densidade máxima em B [1].

É possível ainda construir um algoritmo ainda mais rápido. Tomemos a contrapositiva da Proposição 4.2.1: "Então $B \not\subseteq X$ se, e só se, existe uma **V**-casca de B que não é **V**-dominada por nenhum ponto de X." Ou seja, caso $\Delta_{B_h}(\zeta) > \Delta_X(\zeta)$ para alguma casca ζ de B_h, então $B_h \not\subseteq X$ (Prop. 3.3.3). Com base nestas observações, é possível se conhecer de antemão, algumas posições h para as quais $B_h \not\subseteq X$, ou seja, $h \notin \varepsilon_B(X)$. Estas posições h não precisam serem avaliadas pelo algoritmo de erosão direta.

O algoritmo de erosão direta efetua, basicamente, uma busca do elemento estruturante sobre a imagem de entrada, realizando eventuais saltos sobre os pontos de X, também chamados de *jumps*, assim como em técnicas de *string matching*. Segue o teorema que mostra as condições para que um *jump* possa ser executado:

Teorema 4.2.4 *Este teorema é também conhecido como* ***teorema do jump-miss****, ou seja, saltos por erros.*

Sejam $X, B \subseteq \mathbb{Z}^n$ e $V = \mathcal{V}^{\leftarrow}_{(1,e)} \subseteq \mathbb{Z}^n$ um conjunto de adjacência. Seja $\zeta \in \mathcal{Z}^V(B)$. Se $\Delta_X^V(\zeta) < \Delta_B^V(\zeta)$, então existem pelo menos $\Delta_B^V(\zeta)$ pontos $h \in \mathbb{Z}^n$ tais que $B_h \not\subseteq X$.

Prova: *Seja $k \in V^t$. Segundo a hipótese, temos que $\Delta_X^V(\zeta) < \Delta_B^V(\zeta)$, ou seja, podemos dizer que existe um inteiro positivo α tal que $\Delta_X^V(\zeta) + \alpha = \Delta_B^V(\zeta)$. Observe que dado um conjunto V de adjacência direcional anterior, por definição $\Delta_X^V(\zeta) \geq 0$ e para todo inteiro $i \in [-\Delta_X^V(\zeta), \alpha]$ temos que o ponto $\zeta + i \cdot k$ tem V-densidade $0 \leq \Delta_X^V(\zeta + i \cdot k) \leq \Delta_X^V(\zeta) + i$, visto que a razão incremental da Transformada da Densidade é 1 (ver definição de **V**-densidade). Logo, podemos deduzir que $0 \leq \Delta_X^V(\zeta + i \cdot k) \leq \Delta_B^V(\zeta)$. Além disso, pela Prop. 4.2.2, para qualquer i temos que $\Delta_B^V(\zeta) = \Delta_{B_{i \cdot k}}^V(\zeta + i \cdot k)$, fazendo com que $0 \leq \Delta_X^V(\zeta + i \cdot k) \leq \Delta_{B_{i \cdot k}}^V(\zeta + i \cdot k)$. Sendo assim, temos que $\Delta_X^V(\zeta + i \cdot k) < \Delta_{B_{i \cdot k}}^V(\zeta + i \cdot k)$ para todo $-\Delta_X^V(\zeta) \leq i < \alpha = \Delta_B^V(\zeta) - \Delta_X^V(\zeta)$. Conforme observado anteriormente na Propriedade 4.2.3, como $\Delta_X^V(\zeta + i \cdot k) < \Delta_{B_{i \cdot k}}^V(\zeta + i \cdot k)$ então $B_{i \cdot k} \not\subseteq X$, para todo i. Basta então enumerar todos os possíveis valores de i, que é $\Delta_B^V(\zeta)$. Logo, existem* ***pelo menos*** *$\Delta_B(\zeta)$ pontos $h = i \cdot k$ em $\mathcal{Z}$ tais que $B_h \not\subseteq X$. Note que destes pontos h, α deles ocorrem*

posteriormente a origem de B *e* $\Delta_X^V(\zeta)$ *deles ocorrem anteriormente.* $\diamond$

Além deste teorema nos mostrar a existência de pontos $h \in \mathbb{Z}^n$ para os quais $B_h \not\subseteq X$, sua demons-tração exibe-os explicitamente, ou seja, os $\alpha = \Delta_B^V(\zeta) - \Delta_X^V(\zeta)$ pontos posteriores a origem e os $\Delta_X^V(\zeta)$ anteriores a esta. O vetor k é o próprio vetor que estabelece a ordem da busca para encontrar os $\zeta \in \mathcal{Z}(B)$ tais que $\Delta_{B_h}(\zeta) \leq \Delta_X(\zeta)$, realizando eventuais *jumps*.

O Teorema 4.2.4 fundamenta o algoritmo de erosão direta (o *jump-miss erosion*) e pode ser acompanhado pela Figura 4.4. Nesta figura, suponha que o ponto ζ_2 do elemento estruturante B seja o próprio ponto cinza na imagem de entrada X. Note que $1 = \Delta_X^V(\zeta_2) < \Delta_B^V(\zeta_2) = 4$. Sendo assim, nas posições $p_0 = \zeta_2 + (0,-1)$, $p_1 = \zeta_2 + (0,0)$, $p_2 = \zeta_2 + (0,1)$ e $p_3 = \zeta + (0,2)$ (posições marcadas com $\times$) temos que $\Delta_X^V(p_i) < \Delta_{B_{(0,i-1)}}^V(\zeta)$, para $i \in [1..3]$. Com isso, encontramos 4 posições nas quais o elemento estruturante B não está contido em X. Esta mesma figura exemplifica a Propriedade 4.2.3, isto é, cascas de B com V-densidades menores que os respectivos pontos em X resulta que $B \subseteq X$.

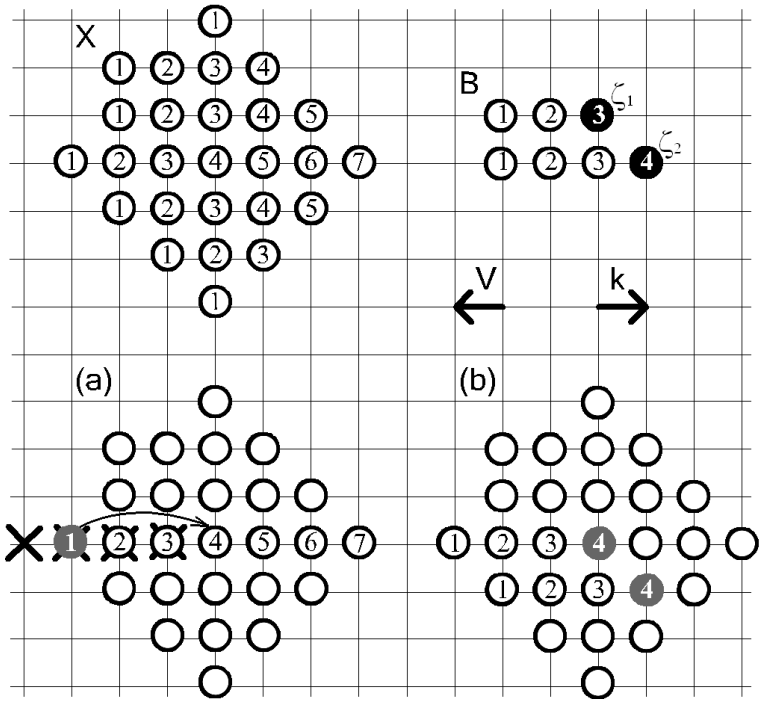


Figura 4.4: As imagens acima correspondem a imagem de entrada X, elemento estruturante B com suas cascas indicadas (pontos em preto), um conjunto de vetores V e o vetor k citados no Teorema 4.2.4. (a) Posições cuja casca ζ_2 de $\mathcal{Z}^V(B)$ não se encaixa no ponto em cinza de X, de acordo com o teorema, onde a seta indica o *jump*. (b) Algumas posições cujas as cascas de $\mathcal{Z}^V(B)$ se encaixam em X.

Usualmente, o conjunto V utilizado pelo algoritmo de erosão direta é um conjunto $\mathcal{V}^{\leftarrow}_{(1,e)}$ horizontal. Em algumas variações do algoritmo de erosão direta, opta-se ordenar as cascas do elemento estruturante em ordem decrescente por densidade. Tal ordem, possibilita avaliar primeiramente as cascas do elemento estruturante de maior densidade, aumentando a chance de pulos maiores em menos comparações.

O algoritmo de erosão **Direct_Erosion**(X, B, $\mathcal{V}^{\leftarrow}_{(1,e)}$) recebe um elemento estruturante B, uma imagem binária X e o conjunto de vetores $\mathcal{V}^{\leftarrow}_{(1,e)}$, e devolve $\varepsilon_B(X)$. Na maioria dos casos, o conjunto $\mathcal{V}^{\leftarrow}_{(1,e)}$ tem orientação horizontal. O vetor de sentido de busca k é o único vetor em $(\mathcal{V}^{\leftarrow}_{(1,e)})^t$.

A lista de prioridades L deve manter o eixo de percurso da busca como o de menor prioridade. No caso de matrizes multidimensionais, o laço respectivo a tal eixo cartesiano deve ser o mais interno.

Algoritmo 5: Direct_Erosion(X, B, $\mathcal{V}^{\leftarrow}_{(1,e)}$) {

```
1:    Z^{V←(1,e)}(B) ← setDensityAndShell(B, V←(1,e));
2:    Execute setDensity(X, V←(1,e));
3:    ε_B(X) ← ∅;
4:    h* ← o menor^L ponto de X;
5:    Seja k o vetor de (V←(1,e))^t;
6:    Para todo h ∈ X em ordem crescente (X, <^L) faça {
7:        Se (h ≥^L h*) então {
8:            Se (∃ζ ∈ Z^{V←(1,e)}(B_h) : Δ_{B_h}^{V←(1,e)}(ζ) > Δ_X^{V←(1,e)}(ζ)) então {
9:                α ← (Δ_{B_h}^{V←(1,e)}(ζ) − Δ_X^{V←(1,e)}(ζ))
10:               h* ← h + α · k;
              } Senão
11:               ε_B(X) ← ε_B(X) ∪ {h};
          }
      }
12:   Devolva ε_B(X);
}
```

O termo **menor**^{L} (ver Linha 4 deste algoritmo) indica que o elemento é o menor segundo a ordenação lexicográfica que toma L como a lista de prioridades, respectivamente.

A multiplicação de um inteiro α pelo vetor k na Linha 10 é um produto de um escalar α por k.

A condição da Linha 7 implicitamente realiza um *jump*, ignorando os pontos contidos no segmento de h até h^*. Dependendo das densidades de ζ em X e em B, o salto é bastante considerável, uma vez que a cada ponto de X saltado o algoritmo economiza até $|\mathcal{Z}^{\mathcal{V}^{\leftarrow}_{(1,e)}}(B)|$ comparações. Este algoritmo é bastante eficiente quando aplicado sobre imagens de entrada que possuem muitos pontos de baixas

densidades.

Análise de Complexidade

A análise de complexidade deste algoritmo é muito dependente da estrutura de dados. Como neste trabalho optamos utilizar matrizes, sejam D_X e D_B as matrizes que armazenam os conjuntos X e B respectivamente.

Podemos utilizar uma lista ligada para armazenar as cascas de B para efetuar mais eficientemente a comparação da Linha 8. De igual modo, podemos armazenar os pontos de X em uma lista ligada em ordem crescente $(X, <^L)$. Tais listas ligadas podem ser montadas no momento em que se realiza a Transformada da Densidade, por exemplo, economizando custos adicionais em tempo linear (com $|V|$ constante).

Com base nestas otimizações, o consumo de tempo deste algoritmo é no pior caso $\Theta(|D_X|+|D_B|)$ para excutar o pré-processamento das Linhas 1 e 2, $\Theta(|D_{\varepsilon_B(X)}|)$ para inicializar (Linha 3), e montar as listas ligadas, além de $O(|X| \cdot |\mathcal{Z}^{\mathcal{V}^{\leftarrow}_{(1,e)}}(B)|)$ para executar as demais linhas. Porém na prática é bem veloz (ver resultados experimentais no Cap. 5), devido aos saltos realizados. Assim, o consumo de tempo do Direct_Erosion é no pior caso $O(|D_X| + |D_B| + |X| \cdot |\mathcal{Z}^{\mathcal{V}^{\leftarrow}_{(1,e)}}(B)|)$, considerando que $|\mathcal{V}^{\leftarrow}_{(1,e)}| = 1$ é constante.

Uma análise mais detalhada do consumo de tempo do algoritmo Direct_Erosion pode ser acompanhada pela seguinte tabela:

Linha do Algoritmo	**Consumo de Tempo**
1	$O(\|D_B\| \cdot \|V\|)$
2	$O(\|D_X\| \cdot \|V\|$
3	$\Theta(\|D_{\varepsilon_B(X)}\|)$
4	$\Theta(\|D_X\|))$
5	$\Theta(1)$
6–7	$O(\|D_X\|)$
8	$O(\|X\| \cdot \|\mathcal{Z}(B)\|)$
9–11	$O(\|X\|)$
12	$\Theta(1)$
Total	$O(\|D_X\| \cdot \|V\| + \|D_B\| \cdot \|V\| + \|X\| \cdot \|\mathcal{Z}(B)\|)$

Jump-hit Erosion: a Erosão Inversa

A erosão inversa efetua uma busca do elemento estruturante sobre a imagem de entrada, marcando-se previamente as posições h em $\mathbb{Z}^n$ das quais se tem certeza que $B_h \subseteq X$.

Este algoritmo também utiliza o conjunto $\mathcal{V}^{\leftarrow}_{(1,e)} \subseteq \mathbb{Z}^n$ como conjunto de adjacência.

O sentido da busca, neste caso, é contrário ao sentido do aumento das densidades na imagem de entrada. O motivo pelo qual tal percurso é contrário se justifica pela condição necessária do Teorema 4.2.5.

Teorema 4.2.5 *Este teorema é também conhecido como* ***teorema do jump-hit****, ou seja, saltos por acertos.*

Sejam $X, B \subseteq \mathbb{Z}^n$ e V um conjunto de adjacência qualquer. Se para toda casca ζ de $\mathcal{Z}^V(B)$ temos que $\Delta_B^V(\zeta) \leq \Delta_X^V(\zeta)$ então certamente existem $\min_{\zeta \in \mathcal{Z}^V(B)}\{\Delta_X^V(\zeta) - \Delta_B^V(\zeta)\} + 1$ pontos $h \in \mathbb{Z}^n$ tais que $B_h \subseteq X$.

Prova: *Note que assim como na hipótese do Teorema 4.2.4 existem pelo menos $\Delta_B^V(\zeta) - \Delta_X^V(\zeta) - 1$ pontos $h \in \mathbb{Z}^n$ posteriores a alguma casca $\zeta \in \mathcal{Z}^V(B)$ (no eixo de uma coordenada k), tais que $\Delta_{B_h}^V(\zeta) > \Delta_X^V(\zeta)$, podemos concluir, não é difícil exibir e mostrar que existem pelo menos $\Delta_B^V(\zeta) - \Delta_X^V(\zeta)$ pontos $h \in \mathbb{Z}^n$ anteriores a ζ (ver figura 4.5), tais que $\Delta_{B_h}^V(\zeta) \leq \Delta_X^V(\zeta)$, onde a diferença $\Delta_{B_h}^V(\zeta) - \Delta_X^V(\zeta)$ é menor possível (garantia de cobertura). Para cada um destes pontos h temos que $B_h \subseteq X$. Como $\Delta_B^V(\zeta) \leq \Delta_X^V(\zeta)$, certamente $B \subseteq X$. Juntamente com $B \subseteq X$, temos $\min_{\zeta \in \mathcal{Z}^V(B)}\{\Delta_X^V(\zeta) - \Delta_B^V(\zeta)\} + 1$ pontos $h \in \mathbb{Z}^n$ tais que $B_h \subseteq X$.*

O Teorema 4.2.5 embasa o algoritmo de erosão inversa (o *jump-hit erosion*) e pode ser acompanhado pela Figura 4.5. Nesta figura, temos na parte superior um exemplo de uma imagem de entrada X e um elemento estruturante B. Note que o conjunto V é um conjunto de adjacência direcional anterior no sentido horizontal composto por um único vetor indicado na figura como k. Conforme citado no início desta seção, pelo fato do percurso ser contrário, o sentido da busca é o mesmo sentido do único vetor k. A parte inferior se divide em dois casos: (a) Este caso evidencia os resultados mostrados pelo Teorema 4.2.5. Os pontos em preto do elemento estruturante (parte superior) são as V-cascas de $\mathcal{Z}^V(B)$. Note que ao "encaixarmos" as V-cascas de B sobre os pontos em cinza do conjunto X (indicados neste caso), as V-densidades dos pontos em cinza são maiores que as respectivas V-cascas de B projetadas sobre estas. Como a diferença mínima entre estas densidades é 2 $(5-3)$, o Teorema 4.2.5 garante que tais V-cascas de B também se encaixarão nos pontos dentro da região sombreada; (b) Este caso mostra que se o conjunto de cascas é independente quanto à cobertura (ver Propriedade 4.2.6), então podemos realizar saltos por erros, assim como no algoritmo *jump-miss erosion*.

Logo, percorrendo sobre o conjunto X em sentido *decrescente* na ordenação lexicográfica, conseguimos encontrar os pontos $h \in \varepsilon_B(X)$. Os $\beta = \min_{\zeta \in \mathcal{Z}^V(B)}\{\Delta_X^V(\zeta) - \Delta_B^V(\zeta)\} + 1$ pontos h podem ser definidos como $h = \zeta + v$, onde $v_k = -i$, para $0 \leq i \leq \beta - 1$ e para todo $j \neq k$ temos que $v_j = 0$.

Note que o uso do conjunto $\mathcal{V}_{(1,e)}^{\leftarrow}$ para representar a adjacência satisfaz a Propriedade de Independência na Cobertura de Cascas:

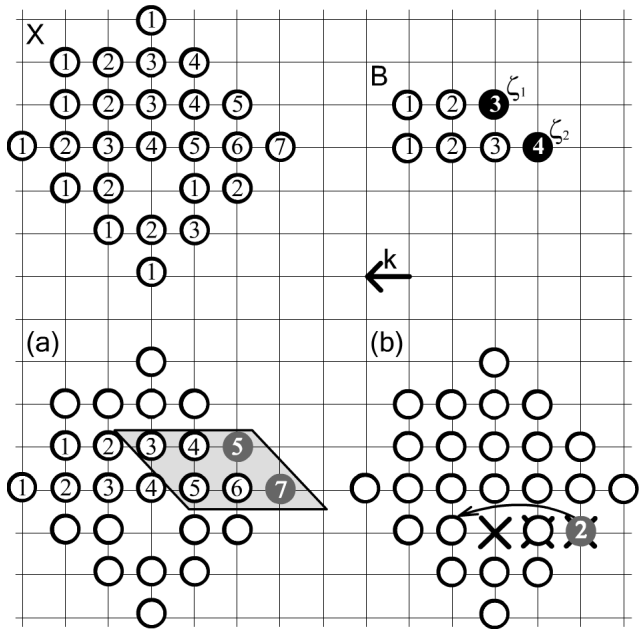


Figura 4.5: As imagens acima correspondem a imagem de entrada X, elemento estruturante B com suas cascas indicadas (pontos em preto), um conjunto de vetores anteriores V. (a) Posições cujas as cascas ζ_1 e ζ_2 de $\mathcal{Z}^V(B)$ se encaixam em X a partir dos pontos em cinza. (b) Posições em que a casca ζ_2 não se encaixa em X a partir do ponto em cinza, pela Propriedade 4.2.6. Tal propriedade só é aplicável em casos em que não há sobreposição de cobertura das **V**-cascas na imagem X.

Propriedade 4.2.6 *Sejam $X, B, V \subseteq \mathbb{Z}^n$. Caso não haja sobreposição de cobertura nas* **V***-cascas de $\mathcal{Z}^V(X)$, isto é, para todo par distinto de* **V***-cascas ζ_1 e ζ_2 de $\mathcal{Z}^V(X)$ tem-se que $COB_X^V(\zeta_1) \cap COB_X^V(\zeta_2) = \emptyset$, então vale que:*

Se existe um $\zeta \in \mathcal{Z}^V(B)$ tal que $0 < \Delta_X^V(\zeta) < \Delta_B^V(\zeta)$ então $B_h \not\subseteq X$, onde $h = i \cdot k$ para algum $k \in V$ e $i \in [1..\Delta_X^V(\zeta)]$ (ver Figura 4.5).

Este resultado segue como corolário do Teorema 4.2.4.

Esta otimização acrescenta ao algoritmo de erosão inversa possibilidades de realizar *jumps*, assim como no algoritmo de erosão direta.

O algoritmo de erosão **Inverse_Erosion**(X, B, $\mathcal{V}^{\leftarrow}_{(1,e)}$) recebe um elemento estruturante B, uma imagem binária X e o conjunto $\mathcal{V}^{\leftarrow}_{(1,e)}$, e devolve $\varepsilon_B(X)$. Tais parâmetros são setados de modo análogo ao algoritmo de erosão direta, bem como o arranjo da lista de prioridades L.

Algoritmo 6: **Inverse_Erosion**(X, B, $\mathcal{V}^{\leftarrow}_{(1,e)}$) **{**

1: $\mathcal{Z}^{\mathcal{V}^{\leftarrow}_{(1,e)}(B)} \leftarrow$ **setDensityAndShell**(B, $\mathcal{V}^{\leftarrow}_{(1,e)}$);
2: `Execute` **setDensity**(X, $\mathcal{V}^{\leftarrow}_{(1,e)}$);
3: $\varepsilon_B(X) \leftarrow \emptyset$;
4: $h^* \leftarrow$ **o maior**^{L} **ponto de** X;
5: `Seja` k **o vetor de** $\mathcal{V}^{\leftarrow}_{(1,e)}$;
6: $inserir \leftarrow falso$;
7: `Para todo` $h \in X$ **em ordem decrescente** $(X, >^L)$ `faça` {
8: `Se` $(h \geqslant^L h^*)$ `então` {
9: `Se` $(\forall \zeta \in \mathcal{Z}^{\mathcal{V}^{\leftarrow}_{(1,e)}}(B_h) : (\Delta^{\mathcal{V}^{\leftarrow}_{(1,e)}}_{B_h}(\zeta) \leq \Delta^{\mathcal{V}^{\leftarrow}_{(1,e)}}_{X}(\zeta))$ `então` {
10: $inserir \leftarrow verdadeiro$;
11: $\beta \leftarrow \min_{\zeta \in \mathcal{Z}^{\mathcal{V}^{\leftarrow}_{(1,e)}}(B_h)} \{\Delta^{\mathcal{V}^{\leftarrow}_{(1,e)}}_{X}(\zeta) - \Delta^{\mathcal{V}^{\leftarrow}_{(1,e)}}_{B_h}(\zeta)\} + 1$;
12: $h^* \leftarrow h + \beta \cdot k$;
13: $\varepsilon_B(X) \leftarrow \varepsilon_B(X) \cup \{h\}$;
} `Senão` {
14: $inserir \leftarrow falso$;
15: $\alpha \leftarrow \Delta^{\mathcal{V}^{\leftarrow}_{(1,e)}}_{X}(\zeta)$;
16: $h^* \leftarrow h - \alpha \cdot k$;
}
17: } `Senão Se` $(inserir)$
18: $\varepsilon_B(X) \leftarrow \varepsilon_B(X) \cup \{h\}$;
}
19: `Devolva` $\varepsilon_B(X)$;
}

O termo **maior**^{L} (ver Linha 4 deste algoritmo) indica que o elemento é o maior segundo a ordenação lexicográfica que toma L como a lista de prioridades, respectivamente.

Assuma os mesmos padrões da análise do algoritmo de erosão direta (estruturas de dados e inicializações).

Análise de Complexidade

O consumo de tempo deste algoritmo é no pior caso $\Theta(|D_X| + |D_B|)$ para excutar o pré-processamento das Linhas 1 e 2 , $\Theta(|D_{\varepsilon_B(X)}|)$ para inicializar (Linha 3), e montar as listas ligadas, além de $O(|X| \cdot |\mathcal{Z}(B)|)$ para executar as demais linhas. Porém na prática é bem veloz, devido aos saltos realizados. Ou seja, a complexidade do algoritmo Inverse_Erosion é a mesma do Direct_Erosion e é no pior caso $O(|D_X| + |D_B| + |X| \cdot |\mathcal{Z}^V(B)|)$, considerando que $|V|$ é constante.

Segue também uma tabela de consumo de tempo detalhada deste algoritmo:

Linha do Algoritmo	**Consumo de Tempo**
1	$O(\|D_B\| \cdot \|V\|)$
2	$O(\|D_X\| \cdot \|V\|$
3	$\Theta(\|D_{\varepsilon_B(X)}\|)$
4	$\Theta(\|D_X\|))$
5–6	$\Theta(1)$
7–8	$O(\|D_X\|)$
9	$O(\|X\| \cdot \|\mathcal{Z}(B)\|)$
10	$O(\|X\|)$
11	$O(\|X\| \cdot \|\mathcal{Z}(B)\|)$
12–18	$O(\|X\|)$
19	$\Theta(1)$
Total	$O(\|D_X\| \cdot \|V\| + \|D_B\| \cdot \|V\| + \|X\| \cdot \|\mathcal{Z}(B)\|)$

Pelo sentido da busca ser contrário sentido da busca do algoritmo Direct_Erosion, este algoritmo é bastante eficiente, quando aplicado sobre imagens de entrada que possuem muitos pontos de alta densidade. Entretanto, este também apresenta bons resultados em imagens com muitos pontos de baixa densidade.

A Figura 4.6 mostra uma simples passo na simulação dos algoritmos de erosão direta e inversa. Observe que a busca das cascas do elemento estruturante B em X neste exemplo ocorre da esquerda para direita na erosão direta e da direita para esquerda na erosão inversa. Nesta figura, observamos dois passos de cada algoritmo apresentados da esquerda para direita, temos que: (a) no algoritmo Direct_Erosion, ao avaliar a casca do elemento estruturante B (transladado conforme mostra a região sombreada) cuja densidade é 4, o algoritmo decide pular 3 posições a frente e continuar a busca. Assim, o algoritmo segue em busca de acertos, conforme mostrado na seqüencia. (b) no algoritmo Inverse_Erosion, a avaliação ocorre no sentido contrário ao aumento das densidades, isto é, da direita para esquerda. Logo no primeiro passo, o algoritmo decide saltar 3 posições das quais se tem garantia que o elemento estruturante B se encaixa na imagem X, uma vez que a menor diferença encontrada entre as densidades das cascas do elemento estruturante e os respectivos pontos em X é $6 - 4 = 2$ (região sombreada na imagem mais à esquerda). A partir daí, o algoritmo encontra uma posição na qual um ponto p de X tem densidade menor que a correspondente casca no conjunto B transladado (região sombreada na imagem mais à direita deste caso). Neste caso, como o conjunto de adjacência satisfaz a propriedade de independência de cascas quanto a cobertura (ver Prop. 4.2.6), o algoritmo pode efetuar um salto de 3 posições, assim como no algoritmo Direct_Erosion. Ambos os algoritmos continuarão avaliando os pontos de outra linha de X, caso um fim de linha é encontrado.

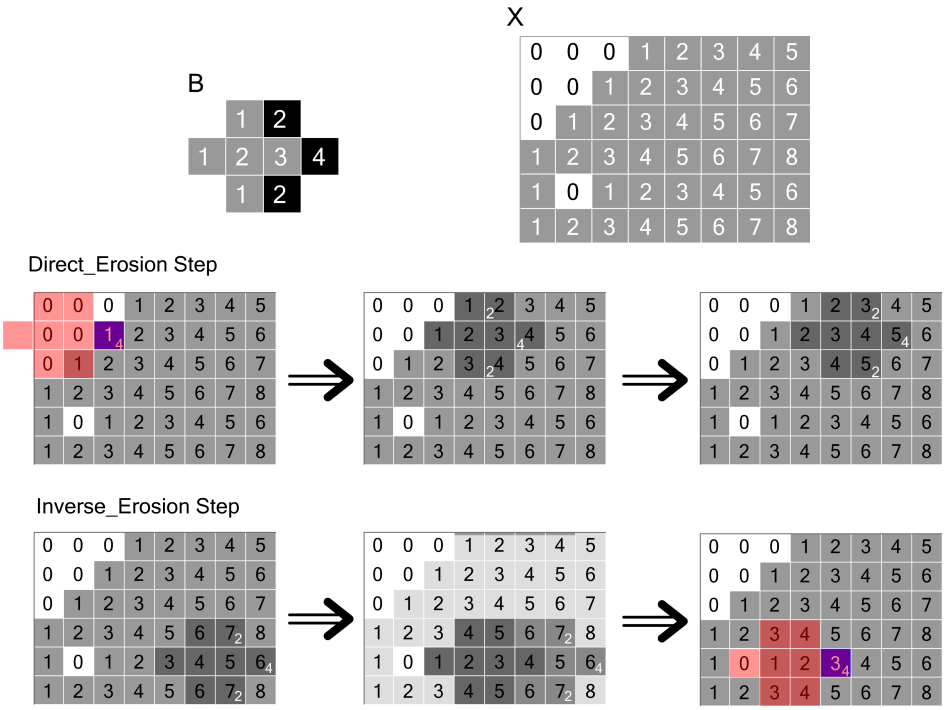


Figura 4.6: Duas simulações de erosão de X por B usando $\mathcal{V}^{\leftarrow}_{(1,e)}$ horizontal para adjacência com o método de erosão direta e inversa. O algoritmo de Erosão Direta (Direct_Erosion) "salta" quando a busca não encontra algum ponto em X e "anda" quando todas as cascas de B são dominada por pontos de X. O algoritmo de Erosão Inversa (Inverse_Erosion) "salta" quando todas as cascas de B são dominada por pontos de X e "salta" quando a busca não encontra algum ponto em X não dominado.

4.2.2 Dilatação usando o conjunto de cascas

Este algoritmo é conhecido como *shell dilation*, ou *dilatação por cascas*. Como próprio nome diz, este algoritmo efetua a dilatação de uma imagem de entrada X por um elemento estruturante B usando basicamente as cascas de X e as cascas de B.

Observe o seguinte fato: Sejam $X, B \in \mathcal{P}(\mathbb{Z}^n)$. Considere que $X = X_1 \cup X_2 \cup \ldots \cup X_N$ e $B = B_1 \cup B_2 \cup \ldots \cup B_M$. Então pela propriedade duplamente distributiva da dilatação (Seção 2.3) temos que $\delta_B(X) = \cup_{1 \leq i \leq N} \cup_{1 \leq j \leq M} \delta_{B_j}(X_i)$.

Por definição, as cascas de um conjunto de cascas qualquer de um conjunto X, juntamente,

cobrem todos os pontos deste (ver Propriedade 3.3.2). Sendo assim, temos que:

$$\delta_B(X) = \bigcup_{\zeta_1 \in \mathcal{Z}^V(B)} \bigcup_{\zeta_2 \in \mathcal{Z}^V(X)} \delta_{COB_B^V(\zeta_2)}(COB_X^V(\zeta_1)),$$

para um conjunto de vetores V qualquer.

Logo, podemos efetuar dilatações das cascas do elemento estruturante pelas cascas da imagem de entrada. Como veremos a seguir, a dilatação dos pontos cobertos por uma casca pelos pontos cobertos por outra pode ser realizado em tempo constante (Prop. 4.2.7).

Propriedade 4.2.7 *Sejam $X, B \subseteq \mathbb{Z}^n$ e V um conjunto de vetores qualquer. Sejam $\zeta_1 \in \mathcal{Z}^V(X)$ e $\zeta_2 \in \mathcal{Z}^V(B)$ duas cascas quaisquer. Então $\delta_{COB_B^V(\zeta_2)}(COB_X^V(\zeta_1)) = COB^V_{\delta_B(X)}(\zeta_3)$, onde $\zeta_3 = \zeta_1 + \zeta_2$ e $\Delta^V_{\delta_B(X)}(\zeta_3) = \Delta^V_X(\zeta_1) + \Delta^V_B(\zeta_2) - 1$.*

Prova: *Segundo a Propriedade 3.3.5 temos que $COB_X^V(\zeta_1) = (m_1 V)_{\zeta_1} = D_1$ e $COB_B^V(\zeta_2) = (m_2 V)_{\zeta_2} = D_2$ onde $m_1 = \Delta^V_X(\zeta_1) - 1$ e $m_2 = \Delta^V_B(\zeta_2) - 1$. Sendo assim, não é difícil ver que $\delta_{COB_B^V(\zeta_2)}(COB_X^V(\zeta_1)) = \delta_{D_2}(D_1) = D_3$ onde $D_3 = ((m_1 + m_2)V)_{\zeta_1+\zeta_2}$. Fazendo $\zeta_3 = \zeta_1 + \zeta_2$ e $m_3 = m_1 + m_2 = \Delta^V_X(\zeta_1) + \Delta^V_B(\zeta_2) - 2$ temos que $\delta_{D_2}(D_1) = ((m_3)V)_{\zeta_3} = COB^V_{\delta_B(X)}(\zeta_3)$, onde é verificado que $\Delta^V_{\delta_B(X)}(\zeta_3) = m_3 + 1 = \Delta^V_X(\zeta_1) + \Delta^V_B(\zeta_2) - 1$ (ver Prop. 3.3.5). Logo, temos que $\delta_{COB_B^V(\zeta_2)}(COB_X^V(\zeta_1)) = COB^V_{\delta_B(X)}(\zeta_3)$, onde $\zeta_3 = \zeta_1 + \zeta_2$ e $\Delta^V_{\delta_B(X)}(\zeta_3) = \Delta^V_X(\zeta_1) + \Delta^V_B(\zeta_2) - 1$.* $\diamond$

Tal resultado sugere que construamos um algoritmo que calcula a dilatação de X por B, utilizando apenas $\mathcal{Z}^V(X)$ e $\mathcal{Z}^V(B)$ para um V qualquer.

A estrutura de dados mais adequada para armazenar os conjuntos $\mathcal{Z}^V(X)$ e $\mathcal{Z}^V(B)$ é a lista ligada. Todavia, uma matriz $D_{\delta_B(X)}$ deve ser utilizada ao final do programa para propagar os pontos cobertos pelas cascas de $\mathcal{Z}^V(\delta_B(X))$ armazenadas, também, em uma lista ligada. O algoritmo **Propagation_Density**$(\mathcal{Z}^V(X), V)$ é usado como subrotina da dilatação por cascas, e estende os pontos cobertos pelas cascas de $\mathcal{Z}^V(X)$, ou seja, devolve o conjunto X.

Algoritmo 7: Propagation_Density($\mathcal{Z}^V(X)$, V) {

```
1:      X ← ∅;
2:      Enquanto Z^V(X) ≠ ∅ faça
3:          Seja ζ uma V-casca de Z^V(X) de V-densidade máxima;
4:          X ← X ∪ {ζ};
5:          Z^V(X) ← Z^V(X) \ {ζ};
6:          Se (Δ^V_X(ζ) > 1) então
7:              Para todo v ∈ V^t faça
8:                  Se (ζ + v ∉ Z^V(X)) então
9:                      Z^V(X) ← Z^V(X) ∪ {ζ + v};
10:                     Δ^V_X(ζ + v) ← 0;
11:                 Se (Δ^V_X(ζ + v) < Δ^V_X(ζ) − 1) então
12:                     Δ^V_X(ζ + v) ← Δ^V_X(ζ) − 1;
13:     Devolva X;
}
```

O custo deste algoritmo depende, basicamente, do acesso aos elementos de densidade máxima do conjunto $\mathcal{Z}^V(X)$. O modo mais apropriado é montar um histograma com as densidades e atualizá-lo a cada iteração. O custo para se montar tal histograma é $O(|\mathcal{Z}^V(X)| + d_{max})$, onde d_{max} é a densidade máxima das cascas de $\mathcal{Z}^V(X)$, que em geral é muito menor que a dimensões de D_X. O custo de atualização e acesso ao histograma é realizado em tempo constante. Supondo que encontrar o ponto de densidade máxima é feito em tempo constante, o consumo de tempo do algoritmo Propagation_Density é $\Theta(|D_X| \cdot |V|)$. Novamente, considerando que $|V|$ é constante, este algoritmo consome tempo linear.

Note que o algoritmo Propagation_Density é exatamente a transformação inversa do algoritmo Shell (ver Seção 3.2).

Segue-se deste modo, um algoritmo de dilatação, denominada **Dilation_Shell**(X, B, V), usando as cascas do elemento estruturante B, da imagem de entrada X e o conjunto de vetores V. Neste algoritmo, não existe nenhuma restrição quanto a escolha do conjunto V.

Algoritmo 8: Dilation_Shell(X, B, V) {

1: $\mathcal{Z}^V(X) \leftarrow$ **Shell**(X);
2: $\mathcal{Z}^V(B) \leftarrow$ **Shell**(B);
3: $\mathcal{Z}^V(\delta_B(X)) \leftarrow \emptyset$;
4: `Para todo` $\zeta_1 \in \mathcal{Z}^V(X)$ `faça`
5: `Para todo` $\zeta_2 \in \mathcal{Z}^V(B)$ `faça` {
6: $\zeta_3 \leftarrow \zeta_1 + \zeta_2$;
7: `Se` $(\zeta_3 \in \mathcal{Z}^V(\delta_B(X)))$ `então`
8: $\Delta^V_{\delta_B(X)}(\zeta_3) \leftarrow \max\{\Delta^V_{\delta_B(X)}(\zeta_3), \Delta_X(\zeta_1) + \Delta_B(\zeta_2) - 1\}$;
`Senão` {
9: $\Delta^V_{\delta_B(X)}(\zeta_3) \leftarrow \Delta_X(\zeta_1) + \Delta_B(\zeta_2) - 1$;
10: $\mathcal{Z}^V(\delta_B(X)) \leftarrow \mathcal{Z}^V(\delta_B(X)) \cup \{\zeta_3\}$;
}
}
11: $\delta_B(X) \leftarrow$ **Propagation_Density**($\mathcal{Z}^V(\delta_B(X))$);
12: `Devolva` $\delta_B(X)$;
}

Análise de Complexidade

O consumo de tempo do algoritmo **Dilation_Shell** é basicamente o mesmo dos algoritmos de erosão para pré-processar os conjuntos X e B, isto é, $\Theta(|D_X| + |D_B|)$, mais o custo do algoritmo de Propagação de densidades ($\Theta(|D_{\delta_B(X)}|)$) somado ao custo da dilatação das cascas propriamente, que é $\Theta(|\mathcal{Z}(X)| \cdot |\mathcal{Z}(B)|)$. Logo, no pior caso o algoritmo de dilatação por cascas é $\Theta(|D_X| + |D_B| + |D_{\delta_B(X)}| + |\mathcal{Z}(X)| \cdot |\mathcal{Z}(B)|)$.

Segue a tabela de consumo de tempo para cada linha deste algoritmo:

Linha do Algoritmo	Consumo de Tempo
1	$O(\|D_B\| \cdot \|V\|)$
2	$O(\|D_X\| \cdot \|V\|$
3	$\Theta(\|D_{\delta_B(X)}\|)$
4	$\Theta(\|\mathcal{Z}(X)\|))$
5–10	$\Theta(\|\mathcal{Z}(X)\| \cdot \|\mathcal{Z}(B)\|)$
11	$\Theta(\|D_{\delta_B(X)}\| \cdot \|V\|)$
12	$\Theta(1)$
Total	$O(\|D_X\| \cdot \|V\| + \|D_B\| \cdot \|V\| + \|X\| \cdot \|\mathcal{Z}(B)\|)$

Note também que o algoritmo que usa RLE [21] para realizar a dilatação é, basicamente, um subcaso de dilatação usando $\mathcal{V}^{\leftarrow}_{(1,e)}$-cascas horizontais.

4.2.3 Conclusão

Neste capítulo foram apresentados dois algoritmos de erosão e um de dilatação.

Os algoritmos de *erosão direta* e *erosão inversa* utilizam algumas técnicas de busca de padrões textuais (p.e. saltos de bits em caso de erro ou acerto) e técnicas de compactação do elemento estruturante (conjunto de cascas). Cada um destes algoritmos deve ser utilizado em casos específicos: o algoritmo de *erosão direta* é bastante eficiente em imagens de entrada que apresentam bastante ruído, ou pontos de baixas densidades, devido à propriedade de efetuar saltos em casos de erros (*miss*); enquanto que o algoritmo de *erosão inversa* é melhor aplicado em imagens de entrada com pontos de altas densidades, devido à propriedade de efetuar saltos em casos de acertos (*hit*).

O algoritmo de *dilatação por cascas* por sua vez, é um algoritmo que calcula a dilatação com base no conjunto de cascas tanto do elemento estruturante, quanto da imagem de entrada.

Para ambos os algoritmos, a escolha do conjunto de adjacência a fim de pré-processar os conjunto, na maioria dos casos, leva em consideração minimizar a cardinalidade do conjunto de cascas do elemento estruturante (ou da imagem de entrada no caso da dilatação). Tal problema é considerado um problema de otimização combinatória, ficando como sugestão para trabalhos futuros.

Estes algoritmos demonstraram um ótimo desempenho em resultados experimentais, conforme serão apresentados no capítulo subseqüente.

Capítulo 5

Resultados Experimentais

Esta seção visa apresentar alguns resultados experimentais dos algoritmos propostos por este trabalho aplicados em imagens bidimensionais, bem como compará-los com outros algoritmos.

Efetuar uma avaliação experimental é uma tarefa bastante complexa. A maioria dos métodos tem características *ad-hoc*, isto é, peculiar a cada aplicação. Entretanto, este trabalho propõe um experimento algorítmico que infere algumas conclusões a respeito dos métodos comparados, de modo que consigamos encontrar um boa alternativa para algoritmos de erosão e dilatação, aplicados aos mais variados casos em aplicações reais.

De antemão, considere que todos os métodos serão submetidos a uma mesma "implementação padrão".

Em tal "implementação padrão", adota-se uma mesma linguagem de programação e mesmas estruturas de dados. A linguagem de programação usada na implementação padrão foi a linguagem C. A estrutura de dados escolhida para armazenar as imagens foi a matriz bidimensional. Uma matriz funciona como uma função indicadora de subconjunto. Ou seja, um elemento $x = (i, j) \in \mathbb{Z}^2$ pertence a uma imagem $X \subseteq \mathbb{Z}^n$, cuja matriz respectiva é M, se, e somente se, $M[i][j] \geq 1$.

Todos os algoritmos para as operações de erosão e dilatação foram executados em uma estação de trabalho *pentium IV* (com 1Gbyte de RAM e 2.4Ghz de CPU) cujo sistema operacional é *Linux*.

Em nossos experimentos usamos *quadrados* Q de dimensão n variando entre 3 e 201 como elementos estruturantes denominados. A escolha destes elementos estruturantes Q são justificadas por se tratarem de elementos estruturantes facilmente decomponíveis (nestes casos, nQ pode ser decomposto por n termos de Q), possuirem contornos bastante reduzido em relação a imagem de entrada (o contorno de um Q tem cardinalidade proporcional a $4n$), por serem elementos estruturantes cujo número de V-cascas é bastante limitado (cada um destes conjuntos Q possui apenas uma V-casca, dependendo do conjunto de adjacência V), por terem um número bem reduzido de "runs" (um Q

possui exatamente n "runs" na compactação RLE) e além disso, por estes pertecerem a classe de formas geométricas mais utilizadas em aplicações reais na MM.

5.1 Imagens de Entrada

As imagens de entrada deste experimento foram categorizadas em imagens *sintéticas*, imagens *naturais* e uma imagem *mista*.

Imagens Sintéticas

As imagens sintéticas foram preparadas a fim de simular a execução de todos os algoritmos, sobre imagens sem buracos (imagens *flat*) com ruído.

Uma imagem sintética ``synth$m$.png'' possui dimensão 512×512, e é composta por uma *Bola Euclidiana* de raio 200 adicionada de um ruído **sal & pimenta** com distribuição uniforme e densidade $D = (10m)\%$ (valor presente no nome da imagem). O valor de D varia de 10% a 100% a uma taxa de 10%, ou seja, $m \in [1..10]$. Veja dois exemplos de imagens sintéticas, uma com 10% de ruído e outra com 50% na Figura 5.1.

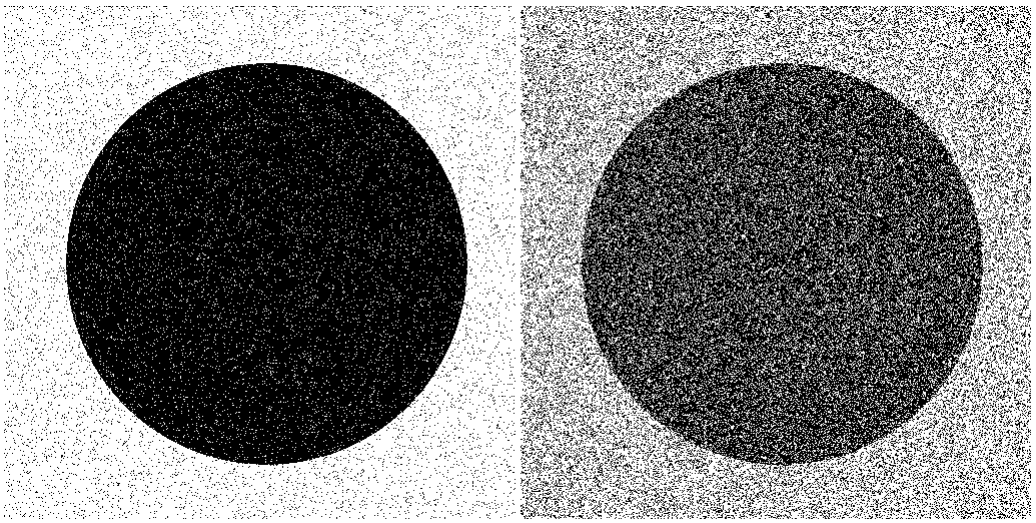


Figura 5.1: À esquerda, a imagem sintética *synth1.png* e à direita, a imagem sintética *synth5.png*

Imagens Naturais

As imagens naturais são imagens encontradas na literatura que se aproximam, um pouco mais, de imagens em aplicações reais.

Como imagens naturais, utilizamos algumas imagens binárias [1] Disponíveis na base de dados de processamento de imagens digitais usadas por Gonzalez [17] em alguns de seus trabalhos. Estas imagens [2] são silhuetas de formas bem conhecidas. Estas imagens podem ser visualizadas na Figura 5.2.

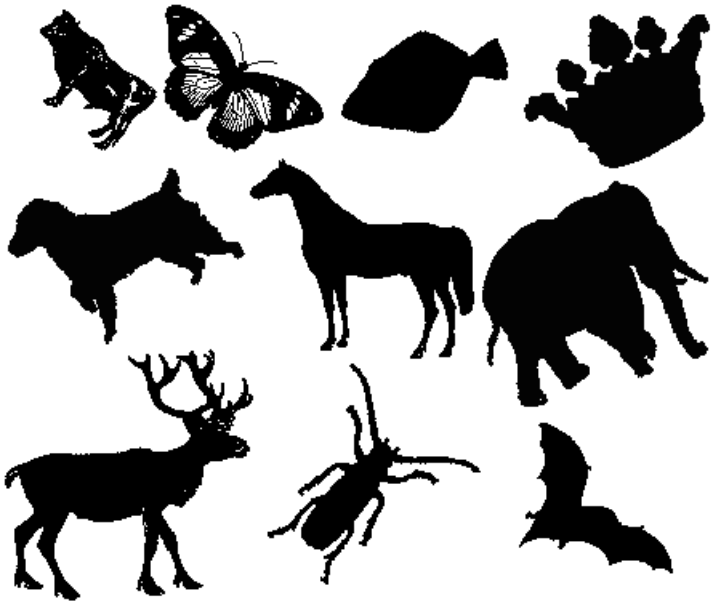


Figura 5.2: Cada silhueta acima corresponde a uma imagem natural utilizada no experimento.

Conforme observamos na figura anterior, o experimento contará com 10 imagens naturais. Em média, uma imagem natural ‘‘`nat`m`.png`’’ possui dimensões em torno de 600×600, onde $m \in [1..10]$.

Imagem Mista

As imagem mista é uma imagem que possui dimensão 2500×2500 e possui quatro quadrantes. Dois quadrantes são compostas por quatro imagens naturais, um quadrante é composto por um ruído sal & pimenta com distribuição uniforme e densidade 100% e o último quadrante ćomposto por linhas

[1] MPEG7 CE Shape-1 Part B

[2] disponíveis em `http://www.imageprocessingplace.com/`

igualmente bem espaçadas e orientadas a 45^o. A Figura 5.3 mostra a imagem mista em um tamanho reduzido.

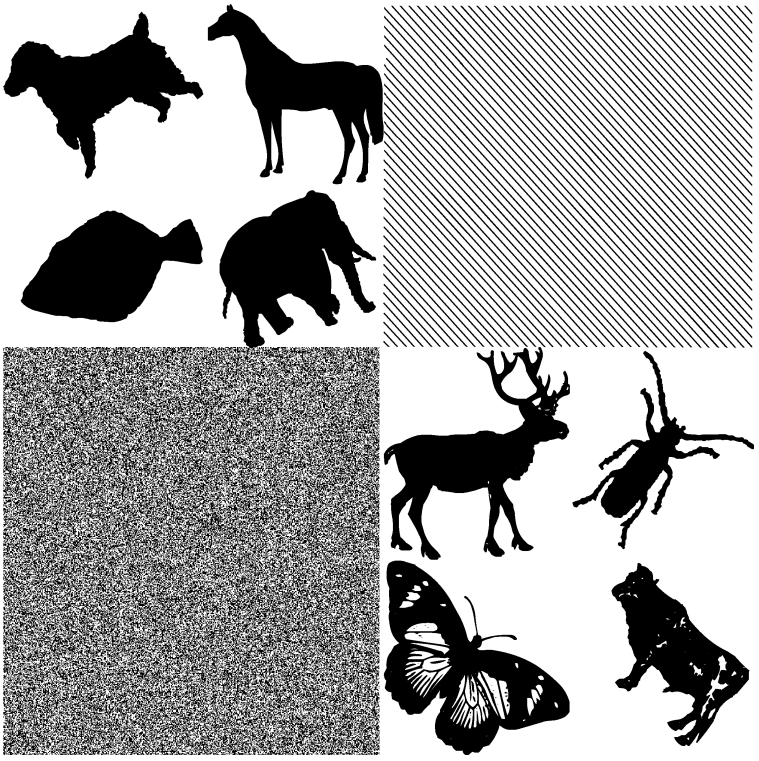


Figura 5.3: A imagem mista de dimensão 2500×2500.

5.2 O Processo

A realização do experimento será feita de forma imparcial, procurando manter os mesmos critérios de avaliação para todos os métodos apresentados, assim como em uma *competição esportiva*. *Duas fases* serão realizadas, a fim de nomear *campeão* um método de erosão e um método de dilatação.

A primeira fase, conhecida como *fase classificatória*, selecionará cinco métodos de erosão e cinco melhores de dilatação mais rápidos segundo uma *média acumulada* (ver adiante) de suas tomadas de tempo. Prosseguindo, o experimento passa para sua segunda fase: a *fase eliminatória*. Nesta fase, os cinco métodos selecionados pela fase anterior (tanto para erosão quanto para dilatação) serão submetidos a um teste final. Este teste usa a imagem mista como imagem de entrada, e nomeia o método campeão.

Os algoritmos de erosão e dilatação utilizados neste experimento serão os seguintes:

Alg. 1: os algoritmos Clássicos.
Alg. 2: os algoritmos que usam BDDs.
Alg. 3: os algoritmos que usam decomposição do elemento estruturante (com BDD).
Alg. 4: os algoritmos que usam vetor de bits.
Alg. 5: os algoritmos que usam RLE.
Alg. 6.1: dilatação que usa contorno da imagem de entrada.
Alg. 6.2: dilatação que usa ambos contornos.
Alg. 7.1: os algoritmos que usam decomposição e o Algoritmo 4.
Alg. 7.2: dilatação que usa decomposição e o Algoritmo 6.1.
Alg. A: o algoritmo de erosão direta (*jump-miss erosion*).
Alg. B: o algoritmo de erosão inversa (*jump-hit erosion*).
Alg. C: o algoritmo de dilatação por cascas (*shell dilation*).

A tomada de tempo dos algoritmos computará os tempos de processamento e pré-processamento juntamente. No caso do Algoritmo C, o conjunto de adjacência utilizado é o conjunto $\mathcal{V}_8^{\leftarrow}$.

Vamos então a descrição da fase classificatória do experimento.

5.2.1 Fase Classificatória

Como o próprio nome diz, esta fase seleciona (classifica) os cinco métodos de erosão e dilatação mais rápidos, a fim de serem testados mais rigorosamente na fase seguinte.

Em todos os testes, utiliza-se quadrados de dimensão $n \in \{3, 5, 7, 11, 15, 25, 51, 75, 101, 151, 201\}$ como elementos estruturantes, bem como utiliza-se as imagens sintéticas e naturais como imagens de entrada.

Nesta fase experimental, os testes com imagens sintéticas são realizados separadamente dos testes com imagens naturais. Para cada um destes testes, geramos uma tabela de *tempo de execução médio* (ver adiante).

Esta fase se divide em duas etapas: as tomadas de tempo médio e a seleção dos métodos finalistas.

Etapa I: *Tomadas de tempo médio*

Nesta etapa, calculamos os tempos de execução de cada algoritmo citado acima, usando todas as imagens sintéticas com $(10m)\%$ de ruído, bem como todas as 10 imagens naturais como imagens de entrada e os quadrados de dimensão n como elementos estruturantes.

No caso de imagens sintéticas, para cada imagem ‘‘`synth`m`.png`’’ com $(10m)\%$ de ruído (ver Seção 5.1) construimos duas tabelas: $\mathcal{T}$-$\mathrm{s}_m^{\varepsilon}$ e $\mathcal{T}$-s_m^{δ}. De igual modo, para cada imagem natural

‘‘`nat`m`.png`’’, construimos uma tabela $\mathcal{T}$-$\mathrm{n}^{\varepsilon}_{m}$ e uma tabela $\mathcal{T}$-n^{δ}_{m}.

Cada entrada de uma tabela qualquer $\mathcal{T}$-$\mathrm{s}^{\varepsilon}_{m}(i,j)$ (ou $\mathcal{T}$-$\mathrm{s}^{\delta}_{m}(i,j)$) corresponde ao tempo de execução do algoritmo i de erosão (ou dilatação) tendo como entrada uma imagem ‘‘`synth`m`.png`’’ qualquer e o quadrado de lado j como elemento estruturante.

Da mesma forma, cada entrada de uma tabela qualquer $\mathcal{T}$-$\mathrm{n}^{\varepsilon}_{m}(i,j)$ (ou $\mathcal{T}$-$\mathrm{n}^{\delta}_{m}(i,j)$) corresponde ao tempo de execução do algoritmo i de erosão (ou dilatação) tendo como entrada uma imagem ‘‘`nat`m`.png`’’ qualquer e o quadrado de lado j como elemento estruturante.

A partir destas tabelas $\mathcal{T}$-$\mathrm{s}^{\varepsilon}_{m}$, $\mathcal{T}$-$\mathrm{n}^{\varepsilon}_{m}$, $\mathcal{T}$-s^{δ}_{m} e $\mathcal{T}$-n^{δ}_{m}, geramos a tabela de tempos médios de execução dos algoritmos de erosão e dos algoritmos de dilatação.

Como dito anteriormente, temos 10 imagens sintéticas e 10 imagens naturais. Sendo assim, as tabelas de tempos médios dos algoritmos de erosão são definidas como:

$$\mathcal{T}\text{-}\mathrm{s}^{\varepsilon}(i,j) = \frac{1}{10}\sum_{k=1}^{10} \mathcal{T}\text{-}\mathrm{s}^{\varepsilon}_{k}(i,j) \quad \text{e} \quad \mathcal{T}\text{-}\mathrm{n}^{\varepsilon}(i,j) = \frac{1}{10}\sum_{k=1}^{10} \mathcal{T}\text{-}\mathrm{n}^{\varepsilon}_{k}(i,j)$$

De igual modo, as tabelas de tempos médios dos algoritmos de dilatação são definidas como:

$$\mathcal{T}\text{-}\mathrm{s}^{\delta}(i,j) = \frac{1}{10}\sum_{k=1}^{10} \mathcal{T}\text{-}\mathrm{s}^{\delta}_{k}(i,j) \quad \text{e} \quad \mathcal{T}\text{-}\mathrm{n}^{\delta}(i,j) = \frac{1}{10}\sum_{k=1}^{10} \mathcal{T}\text{-}\mathrm{n}^{\delta}_{k}(i,j)$$

A Tabela 5.1 nos mostra a tabela $\mathcal{T}$-s^{ε} obtida pelos experimentos aplicados sobre as imagens sintéticas.

Size	*Alg. 1*	*Alg. 2*	*Alg. 3*	*Alg. 4*	*Alg. 5*	*Alg. 7.1*	*Alg. A*	*Alg. B*
3	0.139	0.094	0.124	0.064	0.106	0.056	0.077	0.068
5	0.312	0.096	0.132	0.070	0.107	0.058	0.073	0.068
7	0.559	0.093	0.142	0.078	0.104	0.062	0.072	0.065
11	1.287	0.094	0.157	0.110	0.105	0.068	0.069	0.066
15	2.332	0.095	0.178	0.153	0.107	0.075	0.068	0.065
25	6.351	0.095	0.231	0.331	0.116	0.091	0.068	0.063
51	26.069	0.091	0.360	1.202	0.137	0.136	0.064	0.066
75	55.892	0.095	0.483	2.510	0.157	0.181	0.064	0.065
101	100.469	0.095	0.613	4.533	0.183	0.210	0.062	0.064
151	220.100	0.098	0.862	9.987	0.223	0.307	0.062	0.066
201	381.610	0.098	1.120	17.594	0.272	0.375	0.065	0.070
total	795.120	1.044	4.402	36.632	1.617	1.619	0.744	0.726

Tabela 5.1: Tempo médio de execução (em segundos) dos algoritmos de erosão aplicados sobre as imagens sintéticas.

Todas as tabelas desta seção seguem o mesmo padrão. A primeira linha destas tabelas indica cada algoritmo i de erosão. A primeira coluna destas tabelas indica cada dimensão j do quadrado (elemento

estruturante). As tabelas ainda exibem um total acumulado dos tempos médios dos algoritmos na última linha das tabelas.

A tabela seguinte (Tabela 5.2) corresponde a tabela $\mathcal{T}$-n$^{\varepsilon}$ obtida pelos experimentos, desta vez, aplicados sobre as imagens naturais.

Size	*Alg. 1*	*Alg. 2*	*Alg. 3*	*Alg. 4*	*Alg. 5*	*Alg. 7.1*	*Alg. A*	*Alg. B*
3	0.174	0.095	0.163	0.084	0.110	0.072	0.092	0.090
5	0.391	0.109	0.177	0.094	0.112	0.079	0.096	0.088
7	0.695	0.128	0.199	0.108	0.110	0.084	0.094	0.089
11	1.604	0.189	0.243	0.155	0.113	0.098	0.100	0.090
15	2.914	0.282	0.286	0.225	0.113	0.108	0.100	0.087
25	7.962	0.580	0.390	0.495	0.113	0.136	0.102	0.089
51	32.791	1.596	0.627	1.829	0.118	0.203	0.150	0.095
75	70.589	2.835	0.847	3.814	0.118	0.267	0.156	0.096
101	127.398	7.015	1.029	6.871	0.118	0.335	0.149	0.098
151	281.785	9.779	1.392	15.240	0.121	0.476	0.125	0.104
201	494.067	8.435	1.739	26.900	0.121	0.609	0.107	0.104
total	1020.370	31.043	7.092	55.815	1.267	2.467	1.271	1.030

Tabela 5.2: Tempo médio de execução (em segundos) dos algoritmos de erosão aplicados sobre as imagens naturais.

A Tabela 5.2.1 nos mostra a tabela $\mathcal{T}$-s$^{\delta}$ obtida pelos experimentos sobre as imagens sintéticas.

Size	*Alg. 1*	*Alg. 2*	*Alg. 3*	*Alg. 4*	*Alg. 5*	*Alg. 6.1*	*Alg. 6.2*	*Alg. 7.1*	*Alg. 7.2*	*Alg. C*
3	0.144	0.080	0.120	0.060	0.156	0.216	0.473	0.052	0.235	0.189
5	0.308	0.085	0.131	0.066	0.183	0.303	0.604	0.056	0.271	0.192
7	0.592	0.085	0.141	0.076	0.196	0.442	0.734	0.061	0.301	0.193
11	1.473	0.082	0.158	0.107	0.243	0.851	1.040	0.066	0.349	0.192
15	2.829	0.084	0.178	0.156	0.288	1.515	1.384	0.074	0.401	0.189
25	8.168	0.092	0.231	0.325	0.397	4.152	2.416	0.090	0.534	0.196
51	34.863	0.141	0.360	1.223	0.667	16.973	5.606	0.140	0.868	0.200
75	78.695	0.245	0.491	3.039	0.934	40.061	8.561	0.177	1.166	0.208
101	146.797	0.483	0.611	4.540	1.206	155.189	14.623	0.227	1.489	0.216
151	331.057	1.328	0.870	9.547	1.771	407.214	26.593	0.310	2.251	0.244
201	584.971	5.386	1.132	16.561	2.331	684.436	40.007	0.400	2.786	0.256
total	1189.897	8.091	4.423	35.700	8.372	1311.352	102.041	1.653	10.651	2.275

Finalmente, a Tabela 5.3 nos mostra a tabela $\mathcal{T}$-n$^{\delta}$ obtida pelos experimentos sobre as imagens naturais.

Observe as tomadas de tempo dos algoritmos *Alg. A*, *Alg. B* e *Alg. C*. Estes resultados experimentais confirmam a analise de complexidade dos algoritmos e mostram um bom desempenho dos algoritmos propostos por este trabalho, levando-os a uma posição de destaque em eficiência.

Encontrar todas as tabelas $\mathcal{T}$-s$^{\varepsilon}$, $\mathcal{T}$-n$^{\varepsilon}$, $\mathcal{T}$-s$^{\delta}$ e $\mathcal{T}$-n$^{\delta}$, compreende esta primeira etapa da fase de

Size	*Alg. 1*	*Alg. 2*	*Alg. 3*	*Alg. 4*	*Alg. 5*	*Alg. 6.1*	*Alg. 6.2*	*Alg. 7.1*	*Alg. 7.2*	*Alg. C*
3	0.187	0.103	0.167	0.081	0.111	0.098	0.171	0.076	0.105	0.216
5	0.390	0.141	0.204	0.092	0.114	0.101	0.177	0.081	0.124	0.218
7	0.693	0.198	0.237	0.105	0.113	0.106	0.183	0.088	0.144	0.219
11	1.594	0.367	0.316	0.154	0.113	0.117	0.190	0.099	0.178	0.220
15	2.904	0.621	0.369	0.220	0.115	0.141	0.202	0.111	0.213	0.223
25	7.888	1.541	0.535	0.481	0.119	0.226	0.232	0.134	0.310	0.225
51	32.112	5.418	0.917	1.731	0.127	0.659	0.325	0.202	0.574	0.237
75	68.483	11.000	1.239	3.585	0.129	1.399	0.403	0.258	0.836	0.249
101	121.987	30.171	1.649	6.292	0.139	6.134	0.499	0.333	1.147	0.258
151	263.016	66.532	2.141	14.106	0.146	14.751	0.705	0.455	1.818	0.282
201	446.962	90.030	2.578	24.486	0.168	24.843	0.929	0.611	2.548	0.309
total	946.216	206.122	10.352	51.333	1.394	48.575	4.016	2.448	7.997	2.656

Tabela 5.3: Tempo médio de execução (em segundos) dos algoritmos de dilatação aplicados sobre as imagens sintéticas.

classificação.

Etapa II: *Seleção dos algoritmos finalistas*

Esta etapa simplesmente seleciona dez algoritmos finalistas, sendo que cinco destes são de erosão, e cinco de dilatação.

O método de seleção é extremamente simples: Primeiramente, para cada método i de erosão (ou dilatação), calcule o total acumulado dos tempos médios de execução $\tau^{\varepsilon}(i)$ (ou $\tau^{\delta}(i)$) a partir das tabelas $\mathcal{T}$-s$^{\varepsilon}$ e $\mathcal{T}$-n$^{\varepsilon}$ ($\mathcal{T}$-s$^{\delta}$ e $\mathcal{T}$-n$^{\delta}$). Estes tempos são obtidos através das seguintes fórmulas:

$$\tau^{\psi}(i) = \sum_{\forall m_1} \sum_{\forall j} \mathcal{T}\text{-s}^{\psi}_{m_1}(i,j) + \sum_{\forall m_2} \sum_{\forall j} \mathcal{T}\text{-n}^{\psi}_{m_2}(i,j),$$

onde $\psi = \{\varepsilon, \delta\}$, m_1 é o número de imagens naturais, m_2 é o número de imagens sintéticas e j corresponde a dimensãos dos quadrados (elementos estruturantes).

Finalmente, ordena-se crescentemente as tomadas de tempo τ^{ε} e τ^{δ}, selecionando os cinco métodos mais rápidos de cada tomada. Tais métodos são conhecidos como *métodos finalistas*.

A Tabela 5.4 enumera os métodos de erosão e dilatação finalistas (indicados em negrito).

Algoritmos	***total*** τ^{ε}	***total*** τ^{δ}
Alg. 1	1815.479	2136.113
Alg. 2	32.087	214.213
Alg. 3	**11.494**	**14.775**
Alg. 4	92.447	87.033
Alg. 5	**2.884**	**9.766**
Alg. 6.1	———	1359.927
Alg. 6.2	———	106.057
Alg. 7.1	**4.086**	**4.101**
Alg. 7.2	———	**18.648**
Alg. A	**2.015**	———
Alg. B	**1.756**	———
Alg. C	———	**4.931**

Tabela 5.4: Total acumulado dos tempos médios de execução (em segundos) dos algoritmos de erosão e dilatação aplicados sobre as imagens sintéticas e naturais. Os algoritmos em negrito indicam os algoritmos finalistas.

Conforme elencado, segue a lista dos algoritmos finalistas:

Alg. 3: Erosão e Dilatação com decomposição.
Alg. 5: Erosão e Dilatação com RLE.
Alg. 7.1: Erosão e Dilatação com decomposição e vetor de bits.
Alg. 7.2: Dilatação com usa decomposição e contorno.
Alg. A: Erosão direta (*jump-miss erosion*).
Alg. B: Erosão inversa (*jump-hit erosion*).
Alg. C: Dilatação por cascas (*shell dilation*).

Note que os demais algoritmos foram desclassificados de disputarem a próxima fase.

5.2.2 Fase Eliminatória

O objetivo desta fase é nomear *campeões* dois métodos: um de erosão e um de dilatação.

Para os Algoritmos A e B, o conjunto de adjacência utilizado é o conjunto $\mathcal{V}^{\leftarrow}_{(1,e)} \subseteq \mathbb{Z}^n$ horizontal e para o Algoritmo C o conjunto de adjacência utilizado é o conjunto $\mathcal{V}^{\leftarrow}_{8} \subseteq \mathbb{Z}^n$.

Uma vez que o método é nomeado campeão, não implica necessariamente que tal método é o melhor de todos. Simplesmente, tal resultado sinaliza que o método campeão, em um *caso geral*, é uma boa alternativa para algoritmos de erosão (ou dilatação).

Tal alternativa para um caso geral indica que este algoritmo dependente fracamente da entrada

de dados. Por exemplo, o algoritmo de dilatação por contorno, aplicado sobre imagens *flats*, pode até ser um método mais apropriado, mas em imagens muito ruidosas se torna uma péssima escolha, ou seja, provavelmente, o algoritmo usando contorno não é uma boa alternativa no caso geral.

Por motivos de simplificação, optamos mostrar neste trabalho somente os testes que envolvem os algoritmos finalistas. Outros testes utilizando outros elementos estruturantes e alguns outros resultados complementares podem ser encontrados no site `http://score.ime.usp.br/~dandy`.

Como nesta fase, o número de métodos é bem limitado, e a velocidade destes é relativamente rápida, optamos realizar uma tomada de tempo mais detalhada.

Esta fase do experimento utiliza quadrados de dimensão n variando de 3 a 201 como elemento estruturante. Todavia, tomamos mais tamanhos de elementos estruturante do que em testes anteriores: $n = 2\alpha + 1$, para todo inteiro $\alpha \in [1..100]$. Neste caso, a nossa lista de elementos estruturantes possui 100 quadrados de tamanho ímpar.

Como imagem de entrada, esta fase utiliza a imagem mista, por se tratar de uma imagem bastante grande e consideravelmente completa (muito ruído e muitas regiões planas).

O processo de tomadas de tempo é repetido por 10 vezes para cada elemento estruturante de tamanho n, e ao final, toma-se o tempo médio de execução de cada algoritmo.

Com os dados gerados pelas baterias de testes, optamos por gerar gráficos comparativos entre os cincos algoritmos de cada classe de operador (erosão e dilatação), e ao final, escolher os métodos campeões.

O gráfico da Figura 5.4 mostra os tempos médios (das 10 repetições) de execução dos algoritmos de erosão finalistas usando a imagem mista como imagem entrada e os quadrados de tamanho n como elementos estruturantes. Note que nestas tomadas de tempo, tanto o Algoritmo 3, quanto o Algoritmo 7.1 sofrem influências significativas com o aumento do tamanho do elemento estruturante. Espera-se que estes algoritmos sejam eliminados da lista dos prováveis campeões. No Algoritmo 5 observamos uma razoável *performance* mediante ao crescimento do elemento estruturante, ou seja, o tempo de execução deste algoritmo é pouco influenciado pelo aumento do tamanho do elemento estruturante. Todavia, espera-se que este também seja eliminado da lista dos prováveis campeões, por apresentar tempos de execução superiores aos tempos de execução dos Algoritmos A e B.

Tal fato indica que ambos os algoritmos propostos por este trabalho se destacam por sua eficiência. Tal eficiência é, em parte, justificada para ambos os algoritmos: "*quanto maior a* $\mathcal{V}^{\leftarrow}_{(1,e)}$*-densidade de uma casca do elemento estruturante, espera-se que o algoritmo efetue saltos mais significativos em sua busca*".

Este é o motivo pelo qual os algoritmos de erosão propostos por este trabalho sejam denominados de *jump erosions*, ou *erosões com saltos*.

Sendo assim, um desafio interessante sobre estes algoritmos com saltos é encontrar o conjunto

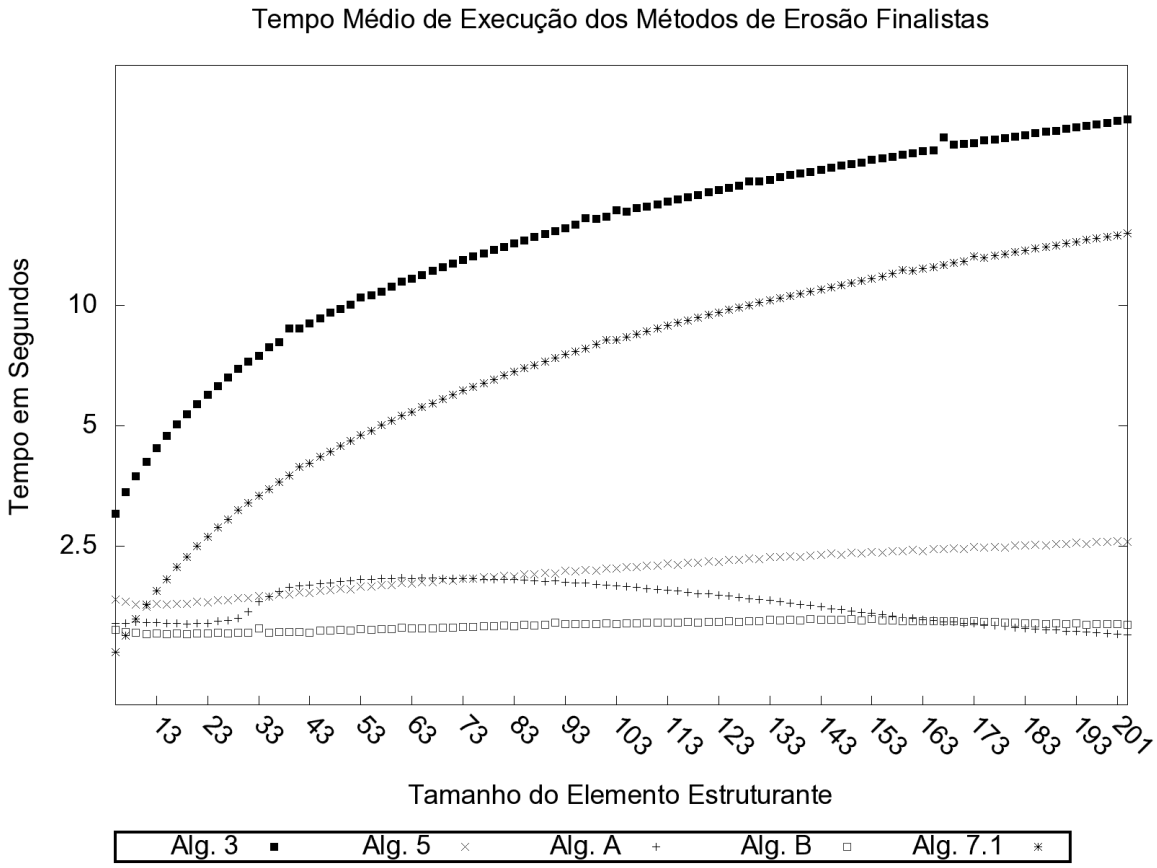


Figura 5.4: Tempo médio de execução (em segundos) dos algoritmos finalistas de erosão após 10 iterações usando como entrada a imagem mista e o quadrado como elemento estruturante.

$\mathcal{V}^{\leftarrow}_{(1,e)}$, ou seja, a escolha do eixo e que maximize o número de saltos destes algoritmos (trabalhos futuros).

O gráfico da Figura 5.5 mostra os tempos de execução dos algoritmos de dilatação finalistas usando o mesmo padrão do gráfico anterior. Note que as tomadas de tempo, tanto do Algoritmo 3, quanto do Algoritmo 7.2 sofrem influências significativas com o aumento do tamanho do elemento estruturante. Espera-se que estes algoritmos também sejam eliminados da lista dos prováveis campeões. Nos Algoritmos 5 e 7.1 observamos um crescimento um pouco menos acentuado em relação aos Algoritmos 3 e 7.2, todavia, observa-se a mesma curva de crescimento. O único algoritmo de dilatação que em suas tomadas de tempo parece não sofrer influências do tamanho do elemento estruturante é o Algoritmo C.

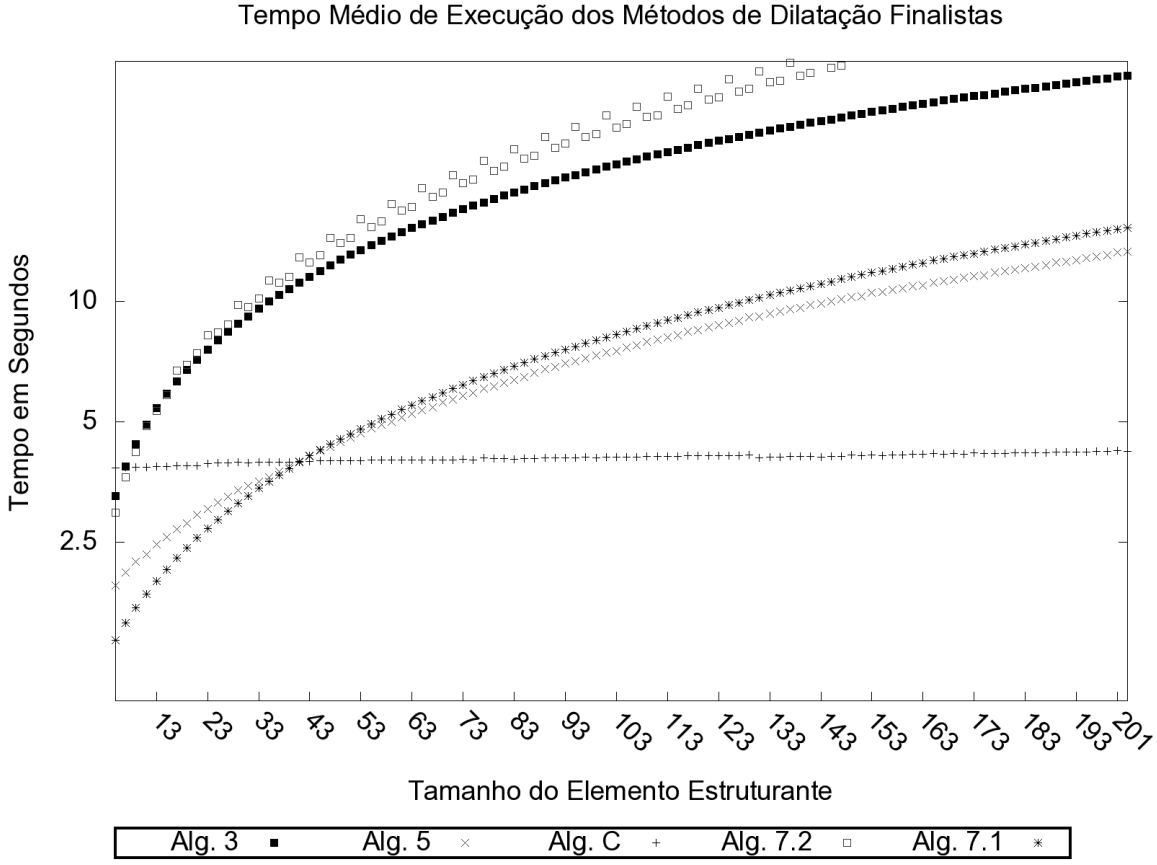


Figura 5.5: Tempo médio de execução (em segundos) dos algoritmos finalistas de dilatação após 10 iterações usando como entrada a imagem mista e o quadrado como elemento estruturante.

Tal fato indica que este algoritmo se destaca por sua eficiência. Este Algoritmo C é o próprio algoritmo de dilatação proposto por este trabalho: a *shell dilation*.

Entretanto, para que o ganho do Algoritmo C seja considerável, o tempo de pré-processamento (e pós-processamento) deve ser bem reduzido. Tal tempo de pré-processamento (e pós-processamento) é completamente dependente do conjunto de adjacência V escolhido como parâmetro deste algoritmo.

Um paradigma neste caso é encontrar um conjunto de adjacência V que minimize o número de V-cascas das imagens de entrada, e ao mesmo tempo seja bem reduzido.

Fica evidente a relação entre o conjunto de adjacência V e a eficiência do Algoritmo C. O conjunto V é responsável pela compressão da imagem de entrada e, principalmente, do elemento estruturante, para se realizar a dilatação. O conjunto $\mathcal{V}_8^{\leftarrow}$ foi escolhido para o Algoritmo C (*shell dilation*) por ser, justamente, o conjunto de adjacência que comprime melhor o elemento estruturante (o quadrado possui apenas uma $\mathcal{V}_8^{\leftarrow}$-casca) e por ser ao mesmo tempo, bastante reduzido (o conjunto $\mathcal{V}_8^{\leftarrow}$ possui apenas 3 vetores).

Encontrar o conjunto V pequeno e que minimiza o número de cascas do elemento estruturante (ou da imagem de entrada) é um desafio interessante para o algoritmo de dilatação, ficando como proposta para trabalhos futuros.

Vamos então a *escolha* dos métodos campeões, um representando a erosão e outro a dilatação. A eleição de um método campeão é bastante simples: *"Um algoritmo de erosão (dilatação) é dito ser campeão, se o total acumulado de seus tempos médios de execução (para cada elemento estruturante) é o menor entre todos os outros métodos de erosão (dilatação)."*

Algoritmos	***total*** τ^{ε}	***total*** τ^{δ}
Alg. 3	1681.368	2897.287
Alg. 5	828.121	2160.225
Alg. 7.1	218.831	836.598
Alg. 7.2	-	762.762
Alg. A	181.100	-
Alg. B	**158.054**	-
Alg. C	-	**406.069**

Tabela 5.5: Total acumulado dos tempos médios de execução (em segundos) dos algoritmos de erosão e dilatação aplicados sobre a imagem mista. Os algoritmos em negrito indicam os algoritmos campeões.

A Tabela 5.5 corresponde ao total acumulado dos tempos de execução destes algoritmos. Note que os Algoritmos **B** e **C** são, respectivamente, os algoritmos de erosão e dilatação com menores tempos acumulados, ou seja, são estes os algoritmos campeões, a saber:

Alg. B: **Erosão inversa (*jump-hit erosion*)**.
Alg. C: **Dilatação por cascas (*shell dilation*)**.

Uma vez encontrados os algoritmos campeões, finalizamos o processo experimental.

Capítulo 6

Conclusão

Conforme mostrado neste trabalho, o pré-processamento é uma boa alternativa para tornar mais rápido as operações morfológicas.

Podemos observar um ganho de velocidade considerável, graças aos pré-processamentos realizados sobre os conjuntos dentro dos métodos propostos por este trabalho.

No caso da erosão, particularmente, o ganho de velocidade está intimamente relacionado à "largura" do elemento estruturante. Ou seja, quanto mais largo o elemento estruturante, maior a densidade de sua casca, e conseqüentemente, maiores possibilidades de *jumps*.

No caso da dilatação, o ganho de velocidade é tanto quanto for maior a compressão dos conjuntos de cascas da imagem de entrada e, principalmente, do elemento estruturante. Esta compressão depende exclusivamente do conjunto de adjacência V. Note que o conjunto V deve ser limitado a um número constante de vetores, caso contrário, os processamentos auxiliares (pré-processamento e pós-processamento) tornam-se custosos para a dilatação. Por exemplo, se executarmos a dilatação de X por B, tomando-se o conjunto de adjacência $V = B$, temos apenas uma V-casca (o próprio elemento estruturante), todavia, com tempo de processamento auxiliar proporcional a $O(|X| \cdot |B|)$. Neste caso, este método é pior do que a dilatação ingênua.

Para se ter uma breve intuição de como escolher um bom conjunto de adjacência, no caso da erosão, observe em qual eixo se concentra a maior "linha digital" contínua. Esta linha determina qual o eixo e do conjunto $\mathcal{V}^{\leftarrow}_{(1,e)}$, ou seja, o próprio eixo na qual esta linha pertence. Já no caso da dilatação, suponha que $B = B_1 \oplus B_2 \oplus \ldots \oplus B_m$. Pode-se, por exemplo, tomar o maior elemento estruturante B_i como o conjunto de adjacência.

Todavia, escolher bons conjuntos de adjacência é um trabalho de otimização combinatória, ficando como sugestão de futuros trabalhos a serem realizados.

Os métodos propostos por este trabalho são tidos como métodos rápidos para suas respectivas tarefas. Dizer que são os mais rápidos métodos de erosão e dilatação em todos os casos não é uma afirmação aceitável. As características específicas da entrada do problema é um fator que impossibilita tal conclusão. Por exemplo, embora o Algoritmo C seja considerado campeão pelo experimento, o Gráfico 5.5 nos mostra que o Algoritmo 7.1 é o mais rápido com elementos estruturantes pequenos.

Além de utilizar estes métodos rápidos aplicados diretamente sobre as imagens binárias, podemos utilizar estes como subrotinas de outras operações morfológicas como no caso de encontrar o contorno direcional de um conjunto qualquer, ou podemos, inclusive, utilizar estes métodos rápidos dentro de outros métodos de dilatação e erosão, como no caso da dilatação por contorno, ou até mesmo, dentro de um método que usa a decomposição de elementos estruturantes.

Parte deste trabalho, sob a forma de um *extended abstract*, de título "Efficient Binary Erosion Algorithm Based on a String-Matching-Like Technique", será divulgado em um congresso de grande porte e de nível internacional, denominado 8^{th} *International Symposium on Mathematical Morphology* (ISMM-2007). Espera-se em breve, divulgar mais resultados deste trabalho em outros congressos e *journals* de grande relevância no meio científico.

Futuramente, espera-se propor novos métodos a partir da Transformada da Densidade e Conjunto de Cascas, tais como aberturas morfológicas e granulometria eficientes, e até mesmo, expandir o conceito de densidades a fim de obter métodos lineares que realizam a dilatação e a erosão morfológica.

Referências Bibliográficas

[1] R. F. Hashimoto A. F. Machado and A. P. Lago, *Efficient binary erosion algorithm based on a string-matching-like technique*, 8th International Symposium on Mathematical Morphology, October 2007, pp. 49–50.

[2] S. B. Akers, *Binary Decision Diagrams*, IEEE Transactions on Computers **C-27** (1978), no. 6, 509–516.

[3] J.C. Alexander and A.I. Thaler, *The boundary count of digital pictures*, **18** (1971), no. 1, 105–112.

[4] C. Arcelli and G. Sanniti di Baja, *A contour characterization for multiply connected figures*, **6** (1987), 245–249.

[5] Ricardo A. Baeza-Yates and Gaston H. Gonnet, *Fast text searching for regular expressions or automaton searching on tries*, Journal of the ACM (1996), no. 43, 915–936.

[6] G. J. F. Banon and J. Barrera, *Minimal Representations for Translation-Invariant Set Mappings by Mathematical Morphology*, SIAM J. Appl. Math. **51** (1991), no. 6, 1782–1798.

[7] ———, *Bases da Morfologia Matemática para a Análise de Imagens Binárias*, IX escola de computação, recife, 1994, This work has been supported by CNPq under contract 300966/90-3.

[8] J. Barrera and R. F. Hashimoto, *Binary Decision Diagrams as a New Paradigm for Morphological Machines*, Mathematical Morphology: 40 Years On (C. Ronse, L. Najman, and E. Decencière, eds.), Computational Imaging and Vision, vol. 30, Springer-Verlag, Dordrecht, 2005, pp. 3–12.

[9] J. Barrera and G. P. Salas, *Set Operations on Closed Intervals and Their Applications to the Automatic Programming of Morphological Machines*, Electronic Imaging **5** (1996), no. 3, 335–352.

[10] G. Birkhoff, *Lattice Theory*, American Mathematical Society Colloquium Publications, Rhode Island, 1967.

[11] Dan S. Bloomberg, *Implementation Efficiency of Binary Morphology*, April 2002.

[12] G. Borgefors, *Distance Transformations on Digital Images*, Computer Vision and Image Processing **51** (1986), no. 34, 344–371.

[13] K. S. Brace, R. L. Rudell, and R. E. Bryant, *Efficient Implementation of a BDD Package*, Proceedings of the ACM/IEEE Design Automation Conference (DAC), ACM/IEEE, 1990, pp. 40–45.

[14] J. Brambor, *Implementation Notes of Binary Dilation and Erosion on 64-bit SH-5 Processor*, October 2002.

[15] R. E. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Transactions on Computers **C-35** (1986), no. 8, 677–691.

[16] Jr Vaughan Pratt Donald Knuth, James H. Morris, *Fast pattern matching in strings*, SIAM Journal on Computing **2** (1977), no. 6, 323–350.

[17] R. Gonzalez and P. Wintz, *Digital image processing*, 2nd ed., 1987.

[18] R. F. Hashimoto and J. Barrera, *A Greedy Algorithm for Decomposing Convex Structuring Elements*, Journal of Mathematical Imaging and Vision **18** (2003), no. 3, 269–289.

[19] R. F. Hashimoto, J. Barrera, and C. E. Ferreira, *A Combinatorial Optimization Technique for the Sequential Decomposition of Erosions and Dilations*, Journal of Mathematical Imaging and Vision **13** (2000), no. 1, 17–33.

[20] H. J. A. M. Heijmans, *Morphological Image Operators*, Academic Press, Boston, 1994.

[21] P. Verbeek I.Young, R. Pevereni and P. Otterloo, *A new implementation for binary and Minkowski operators*, Computer Graphics and Image Processing **17** (1981), 189–210.

[22] A. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, Boston, 1989.

[23] C.Y. Lee, *Representation of Switching Circuits by Binary-Decision Programs*, Bell Systems Technical Journal **38** (1959), 985–999.

[24] Y.T. Liow, *A contour tracing algorithm that preserves common boundaries between regions*, **53** (1991), no. 3, 313–321.

[25] H. M. F. Madeira and J. Barrera, *Incremental Evaluation of BDD-Represented Set Operators*, SIBGRAPI 2000 - XIII Brazilian Symposium of Computer Graphic and Image Processing, IEEE Computer Society, 2000, pp. 308–315.

[26] H. M. F. Madeira, J. Barrera, R. Hirata Jr., and N. S. T. Hirata, *A New Paradigm for the Architecture of Morphological Machines: Binary Decision Diagrams*, SIBGRAPI'99 - XII Brazilian Symposium of Computer Graphic and Image Processing, IEEE Computer Society, November 1999, pp. 283–292.

[27] H. M. F. Madeira and H. Pedrini, *Boolean Decomposition of Binary Image Operators.*, SIBGRAPI, 2002, pp. 313–320.

[28] Ravi Malladi, James A. Sethian, and Baba C. Vemuri, *Shape modeling with front propagation: A level set approach*, IEEE Transactions on Pattern Analysis and Machine Intelligence **17** (1995), no. 2, 158–175.

[29] P.A. Maragos and R.W. Schafer, *Morphological skeleton representation and coding of binary images*, **34** (1986), 1228–1244.

[30] Donald Michie, *Memo Functions and Machine Learning*, Nature (1968), no. 218, 19–22.

[31] L. Piper and J.Y. Tang, *Erosion and Dilation of Binary Images by Arbitrary Structuring Elements Using Interval Coding*, Pattern Recognition Letter **9** (1989), no. 3, 201–209.

[32] P.Zamperoni, *Dilatation und Erosion von konturcodierten Binärbildern*, Microscopica Acta (1980), 245–249.

[33] J. S. Moore R. S. Boyer, *A fast string searching algorithm*, Comm. ACM **20** (1977), 762–772.

[34] L. Robert and G. Malandain, *Fast Binary Image Processing Using Binary Decision Diagrams*, Computer Vision and Image Understanding **72** (1998), no. 1, 1–9.

[35] J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, New York, 1982.

[36] ———, *Image Analysis and Mathematical Morphology. Volume 2: Theoretical Advances*, Academic Press, 1988.

[37] F.Y.C. Shih and Y.T. Wu, *Decomposition of binary morphological structuring elements based on genetic algorithms*, **99** (2005), no. 2, 291–302.

[38] P. Soille, *Morphological Image Analysis*, Springer-Verlag, New York, 1999.

[39] R. M. Haralick Su Chen, *Recursive erosion, dilation, opening, and closing transforms*, IEEE Transactions on Computers **4** (1995), no. 3, 335–345.

[40] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein, *Introduction to algorithms*, MIT Press and McGraw-Hill, 2001.

[41] E. Ukkonen, *On-line construction of suffix trees*, Algorithmica **3** (1995), no. 14, 249–260.

[42] Rein van den Boomgaard and Richard van Balen, *Methods for fast morphological image transforms using bitmapped binary images*, CVGIP: Graphical Models and Image Processing **54** (1992), no. 3, 252–258.

[43] M. Van Droogenbroeck, *Algorithms for openings of binary and label images with rectangular structuring elements*, Mathematical morphology (H. Talbot and R. Beare, eds.), CSIRO Publishing, Sydney, Australia, April 2002, pp. 197–207.

[44] Luc Vincent, *Morphological Transformations of Binary Images with Arbitrary Structuring Elements*, Signal Processing - Image Communication **22** (1991), no. 1, 3–23.

[45] P. Weiner, *Linear pattern matching algorithm*, 14th Annual IEEE Symposium on Switching and Automata Theory, ACM/IEEE, 1973, pp. 1–11.

[46] Seong-Dae Kim Wook-Joong Kim and Kyuheon Kim, *Fast Algorithms for Binary Dilation and Erosion Using Run-Length Encoding*, ETRI Journal **27** (2005), no. 6, 814–817.

[47] J. Xu, *Decomposition of Convex Polygonal Morphological Structuring Elements into Neighborhood Subsets*, IEEE Transactions on Pattern Analysis and Machine Inteligence **13** (1991), no. 2, 153–162.

[48] H.T. Yang and S.J. Lee, *Decomposition of morphological structuring elements with integer linear programming*, **152** (2005), no. 2, 148–154.

[49] X. Zhuang and R.M. Haralick, *Morphological Structuring Element Decomposition*, Computer Vision, Graphics and Image Processing **35** (1986), 370–382.