Capítulo 4

Classificação da TD

Neste capítulo serão classificados os algoritmos clássicos da *transformada de distância* (TD) que podem ser implementados por erosões nos padrões paralelo, seqüencial e por propagação. Estas erosões apresentam melhor eficiência quando usados em decomposições de funções estruturantes. Como resultado desta classificação serão propostos dois novos algoritmos para a *transformada de distância euclidiana* (TDE): um direcional e outro multidimensional.

4.1 Definições

Métricas particulares serão apresentadas nesta seção. Com estas métricas usadas nas funções estruturantes é possível computar erosões morfológicas e obter a TD.

Sejam $x = (x_1, x_2) \in \mathbb{Z} \times \mathbb{Z}$ e $y = (y_1, y_2) \in \mathbb{Z} \times \mathbb{Z}$. $d(x, y)$ é uma *distância* entre x e y, se:

(i) $d(x, y) = d(y, x)$;

(ii) $d(x, y) \geq 0$;

(iii) $d(x,x) = 0$.

Se, além disso, *iv* e *v* abaixo também forem satisfeitas, então d é uma *métrica*.

(iv) $d(x,y) = 0 \Longleftrightarrow x = y$;

(v) $d(x,y) \leq d(x,z) + d(z,y), \forall z \in \mathbb{Z} \times \mathbb{Z}$.

Algumas métricas usuais para $d(x,y)$ são apresentadas a seguir [Bor86, Cui99]:

***City-block*:** $d_4(x,y) = |x_1 - y_1| + |x_2 - y_2|$;

***Chessboard*:** $d_8(x,y) = \max\{|x_1 - y_1|, |x_2 - y_2|\}$;

***Chanfrada A:B*:** $d_{A:B}(x,y) = A * \max\{|x_1 - y_1|, |x_2 - y_2|\} + (B - A) * \min\{|x_1 - y_1|, |x_2 - y_2|\}$, com $0 < A \leq B$ inteiros;

***Euclidiana*:** $d_E(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$.

A *função distância de um pixel x ao conjunto X* é definida como [Ser82]:

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

A *função distância* (denotada $\Psi_d(f)$ e chamada neste texto de *transformada de distância* ou simplesmente TD) é definida como:

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

Esta função atribui a cada pixel de um objeto numa imagem binária o valor da distância mínima ao fundo. Em outras palavras, atribui a cada pixel de um objeto a menor distância entre este pixel e um pixel de fundo.

Serão usadas neste texto imagens com níveis de cinza como um subconjunto dos números inteiros $\mathbb{Z}$. Para a métrica *euclidiana*, será usado o

quadrado da distância euclidiana $(d_E)^2 \in \mathbb{Z}$, para permanecer com imagens inteiras das funções e obter a *transformada de distância euclidiana* (TDE). Assim, serão apresentados algoritmos para $(\text{TDE})^2$ e para obter exatamente a métrica euclidiana, deve-se calcular a raiz quadrada de todos os níveis de cinza gerados pela transformação. Portanto, ao dizer que um algoritmo obtém a métrica euclidiana, na verdade está obtendo o quadrado da métrica euclidiana.

4.2 Tipos de classificações da TD

A TD pode ser classificada de várias formas [Cui99]: pela métrica mais pobre (*city-block*) à mais exata (*euclidiana*), pela complexidade, pela eficiência, pela ordem de varredura, etc. Na tese do Cuisenaire [Cui99], existe uma classificação da TD da métrica mais pobre à mais exata, sendo divididas como *TD euclidiana aproximada* e *TD euclidiana exata*, respectivamente.

Na introdução desta tese, Seção 1.1, foi feito uma classificação da TD quanto à métrica usada, podendo ser *transformada de distância chanfrada* (TDC), incluindo as métricas onde a decomposição da função estruturante é constante. Exemplos de métricas que produzem a TDC: *city-block*, *chessboard*, *octagonal*, *chanfrada 3:4*, *chanfrada 5:7:11*, etc. A métrica mais natural para computar a distância na maioria das aplicações é a *euclidiana*, principalmente em função de sua propriedade de rotação invariante. Porém, devido a dificuldades de construir algoritmos eficientes para a *transformada de distância euclidiana* (TDE), muitos pesquisadores desenvolveram algoritmos aproximados (TDEA).

Uma classificação da TD quanto à ordem de varredura dos pixels na imagem foi feita na Seção 1.2. Três padrões de algoritmos foram definidos: paralelo, seqüencial e por propagação. Estes padrões estão descritos nas Subseções 4.3.1, 4.3.2 e 4.3.3, respectivamente.

Finalmente, na Seção 1.3, a TD por propagação foi subdividida em ou-

tros quatro tipos: propagação vetorial, propagação ordenada, propagação (ou propagação paralela) e geométrico. Vale observar que alguns algoritmos pertencem a mais de um tipo apresentado, como o do Ragnemalm [Rag92] e o do Eggers [Egg98] que são por propagação vetorial e também por propagação paralela. Mais informações para esta classificação são apresentadas a seguir.

TD por propagação vetorial

Os algoritmos da TD por propagação vetorial são difíceis de relacionar com a morfologia matemática, pois são decompostos em informações vetoriais do plano cartesiano discreto.

Cuisenaire [Cui99] chamou de *TD por propagação vetorial* os algoritmos que em vez de manipularem as distâncias (por exemplo, $d_4(x, y)$, $d_8(x, y)$, ...), manipulam os vetores distância (por exemplo, $(|x_1 - y_1|, |x_2 - y_2|)$, onde $x = (x_1, x_2) \in \mathbb{Z} \times \mathbb{Z}$ e $y = (y_1, y_2) \in \mathbb{Z} \times \mathbb{Z}$). O primeiro a implementar algoritmos seqüenciais para esta classe de TD foi Danielsson [Dan80]. Estes algoritmos não resultam na TDE, porém são eficientes (foram inspirados nos algoritmos seqüenciais definidos por Rosenfeld e Pfaltz [RP66]).

Ragnemalm [Rag92] modificou os algoritmos seqüenciais da TD definidos por Danielsson. Ele definiu um novo algoritmo por propagação vetorial que encontra a TDE, onde utiliza duas estruturas de filas para armazenar os pixels de propagação e várias condições de propagação. Para cada direção de propagação existe um teste, gerando um algoritmo complexo.

Eggers [Egg98] melhorou o algoritmo de Ragnemalm diminuindo os vários testes de direção, melhorando a velocidade do algoritmo de propagação, mas aumentando a complexidade da codificação.

Os trabalhos de TD por propagação vetorial, como os de Danielsson [Dan80], Ragnemalm [Rag92] e Eggers [Egg98], não serão reescritos por erosão morfológica pois necessitam manipular informações vetoriais e as erosões definidas neste texto são as clássicas, ou seja, manipulam distâncias e não vetores. Porém, inspirado no trabalho do Eggers, será proposto um

algoritmo que usa erosões e as informações de direção de propagação, como será apresentado na Seção 4.4.

O algoritmo do Danielsson [Dan80] e a primeira parte do algoritmo do Cuisenaire [Cui99] não calculam a TDE devido a não conexidade do diagrama de Voronoi no caso discreto. O *diagrama de Voronoi* divide um plano formados por polígonos desconexos e as uniões destes polígonos formam novamente o plano. A não conexidade do diagrama de Voronoi sempre ocorre próximo aos cantos dos polígonos de Voronoi, descrito inicialmente por Danielsson e corrigido por Cuisenaire na segunda parte do seu algoritmo da TDE. Yamada [Yam84] foi o primeiro a propor um algoritmo exato (que também é por propagação) que resolve este problema. A Figura 4.1 ilustra duas imagens onde ocorrem o erro nos algoritmos definidos por Danielsson. Ambas imagens contém três pixels ($p1$, $p2$ e $p3$) de valores zeros e os restantes com valores uns. A imagem à esquerda ilustra a métrica *city-block* e a imagem à direita a métrica *chessboard*. A região ao redor do pixel $p2$ é desconectado do pixel q pelas regiões geradas pelos pixels $p1$ e $p3$, onde $r1$ pertence à região (polígono) propagada a partir de $p1$ e $r2$ pela região propagada a partir de $p3$. Assim, na figura à esquerda q deveria receber $\sqrt{8}$ e não $\sqrt{9}$ ($q - p1 = (3, 0)$, $q - p2 = (2, 2)$, $q - p3 = (0, 3)$). Na figura à direita q deveria receber $\sqrt{144 + 25}$ e não $\min\{\sqrt{169 + 1}, \sqrt{121 + 49}\}$ ($q - p1 = (13, 1)$, $q - p2 = (12, 5)$ e $q - p3 = (11, 7)$). A TDE (como o diagrama de Voronoi) não é uma propriedade local, a informação de vizinhança de um pixel não é propagada para pixels distantes. Portanto esse problema da não conexidade ocorre pois são propagadas informações locais, vizinho-por-vizinho.

TD por propagação ordenada

Os algoritmos por *propagação ordenada* são melhores modelados como o problema de caminho mais curto usado em teoria dos grafos, implementados usando fila hierárquica. Não é possível modelar este problema por erosões morfológicas.

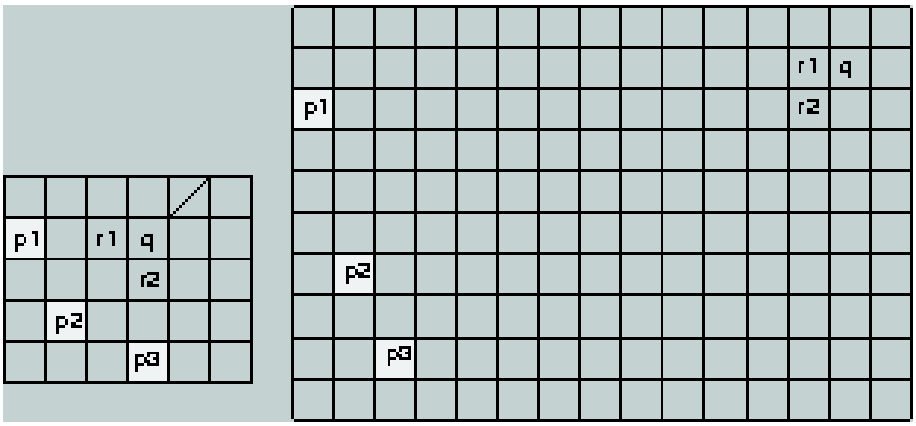


Figura 4.1: Erros ocorridos nos algoritmos de Danielsson: para métrica *city-block* (esquerda) e *chessboard* (direita) (fonte [Cui99]).

Sharaiha e Christofides [SC94] propuseram uma aproximação *baseada em grafos* para a TD. A TD baseada em grafos pode ser reduzida ao problema de floresta de caminho mínimo, onde as raízes das árvores são determinadas pelos pixels de fronteira dos objetos da imagem. Sharaiha e Christofides descreveram uma TD *chanfrada* baseada em grafos e apontaram as principais vantagens de usar a teoria dos grafos em vez de usar a aproximação pixel-por-pixel usada em processamento de imagens. A saber: o tamanho do grafo é normalmente menor que o tamanho da imagem; cada pixel é normalmente processado uma única vez; o algoritmo pode ser estendido para dimensões maiores e para diferentes grades topológicas (por exemplo, grade hexagonal).

Outro trabalho importante da TD por propagação ordenada foi definido por Lotufo *et. al.* [LFZ02], onde definiu o algoritmo *IFT* (*Image Floresting Transform*). A *IFT* tem como entrada uma imagem em níveis de cinza, uma outra imagem com marcadores e uma função estruturante para determinar a vizinhança. Como saída, o algoritmo retorna uma imagem de custos (para algumas entradas especiais esta imagem de custos é exatamente a TD), uma imagem representando a propagação de cada marcador, ou *watershed*, e o caminho mínimo até o marcador mais próximo.

Um estudo de *transformações baseadas em grafos* pode ser encontrado em Zampirolli [Zam97], onde se criou um conjunto de ferramentas baseadas em grafos na *toolbox MMach* [BBL94], usando o ambiente *KHOROS*, inclusive incluindo a TD por propagação ordenada.

TD por propagação

Os algoritmos da TD por *propagação* (ou *propagação paralela*) têm relações diretas com erosões morfológicas e serão estudados nas próximas seções.

Os trabalhos de Ragnemalm [Rag92], Eggers [Egg98] e Yamada [Yam84] apresentados anteriormente no tipo de propagação vetorial também são por propagação paralela. A chave destes algoritmos é: usar duas estruturas de armazenamento para o mapa de custos (*custo_velho* e *custo_novo*); propagar todos os pixels em paralelo atualizando *custo_novo* apenas com os dados de *custo_velho*; depois que tudo foi atualizado, trocar *custo_novo* por *custo_velho* e fazer outra propagação paralela; continuar fazendo isto até não existir mais propagação. Estes algoritmos não são baseados explicitamente em erosão, porém suas implementações são bastante semelhantes ao algoritmo da TDE usando erosão por propagação apresentado na Subseção 4.3.3. O trabalho de Eggers [Egg98] propôs várias melhorias ao trabalho de Ragnemalm [Rag92] fazendo um algoritmo que realiza menos acessos aos pixels. A idéia é fazer uso da direção da propagação para restringir a varredura da vizinhança apenas aos pixels que podem ser modificados. Usando as informações contidas no algoritmo do Eggers é possível alterar o algoritmo da TDE utilizando erosão por propagação para que faça uso da direção da propagação, como apresentado na Seção 4.4.

Outro trabalho importante para esta tese foi feito por Vincent [Vin92], que implementou a TD por propagação a qual está apresentada no Algoritmo 11, Subseção 4.3.3. Este algoritmo da TD usa uma estrutura de fila e poderia também ser usado para calcular a TDE, porém há a necessidade

de trocar a função estruturante a cada erosão e fazer cópia de imagens. Isto poderia ser feito colocando na fila um *token* indicando o fim de cada erosão. A cada retirada do *token*, seria trocado a função estruturante. A reescrita do Algoritmo 11 é apresentada no Algoritmo 13, porém sem o uso de fila.

TD geométrica

Os algoritmos *geométricos* para a TD usando intersecção de parábolas formam o último tipo de TD por propagação. Como exemplo, Saito e Toriwaki [ST94] mostraram que a TDE é separável. Este problema é exatamente o que faz a morfologia matemática, quando o problema é decomposto em elementos unidimensionais. Nesta linha existem vários outros artigos: como o que valida o espaço contínuo [EBS01]; o algoritmo definido por Meijster e Roerdink [MR00], um dos mais eficientes reportados na literatura; e também o proposto neste documento, veja Seção 4.5.

4.3 Padrões de algoritmos da TD usando erosões

Shih e Mitchell [SM92] mostraram que a TD pode ser computada através de uma erosão morfológica por uma função estruturante dada pelo negativo da distância à origem. Mais tarde, Huang e Mitchell [HM94] chegaram em uma computação eficiente para a TDE decompondo a função estruturante euclidiana por uma seqüência de funções estruturantes diferentes de tamanho 3×3.

O trabalho de Mitchell *et. al.* [SM92, HM94] abriu uma porta para estudar e classificar a diversidade de algoritmos da TD disponíveis na literatura. Devido ao fato de existirem muitos caminhos para implementar eficientemente a erosão morfológica, Zampirolli e Lotufo [ZL00] propuseram uma classificação de algoritmos da TD, onde são analisados quais algoritmos da

erosão e quais esquemas de decomposição de função estruturante foram utilizados na literatura.

4.3.1 TD paralela

TD usando erosão paralela

Shih e Mitchell [SM92] propuseram um algoritmo para a TD usando a erosão morfológica aplicada em uma imagem binária f por uma função estruturante b_G.

Veja na Figura 4.2 um exemplo da TD usando erosão para o caso unidimensional. Assim, seja a seguinte equação para a TD:

$$\Psi_d(f) = \varepsilon_{b_G}(f). \tag{4.1}$$

O valor na origem de b_G é zero e nos outros pixels é dado pelo negativo da distância à origem, isto é,

$$b_G(x) = -d(x, \mathcal{O}), x \in \mathbb{E} \oplus \mathbb{E}, \tag{4.2}$$

onde $\mathcal{O} = (0, 0)$. O tipo da TD (*city-block*, *chessboard*, *octagonal*, *chanfrada*, *euclidiana*, etc) vai depender da métrica utilizada pela função estruturante.

Uma propriedade da erosão específica para a TD é ser *idempotente*, isto é, ao aplicar a erosão por b_G novamente, o resultado não é modificado [LT00][1]:

$$\varepsilon_{b_G}(\varepsilon_{b_G}(f)) = \varepsilon_{b_G}(f). \tag{4.3}$$

Esta propriedade é útil, por exemplo, quando se deseja trabalhar com a

[1]Um trabalho interessante que estuda as condições para que dilatações e erosões em espaço-escala morfológico sejam idempotentes foi realizado por Leite e Teixeira [LT00]. Este trabalho será comentado nas conclusões desta tese como trabalhos futuros para a TD.

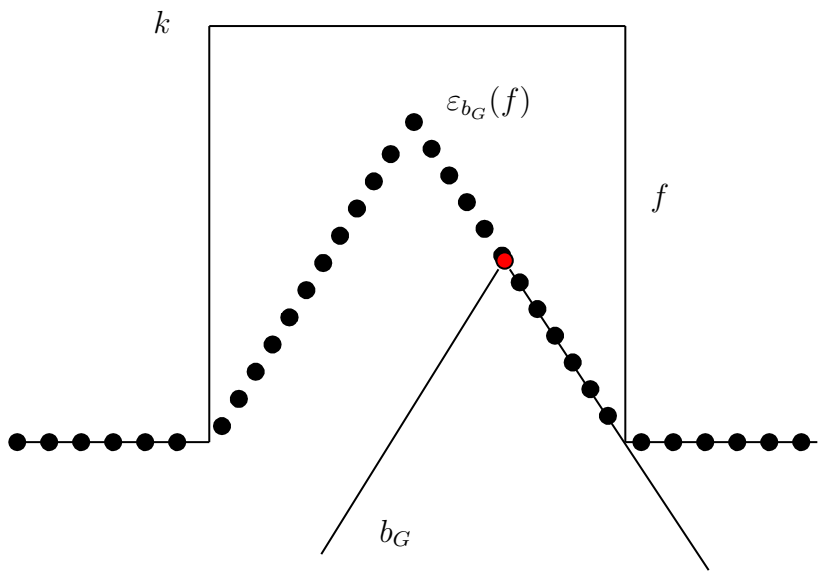


Figura 4.2: TD usando erosão unidimensional da imagem f por b_G.

decomposição da função estruturante b_G para obter algoritmos eficientes e estáveis, como serão apresentados a seguir.

Pelos estudos de decomposição paralela de função estruturante apresentados na Subseção 3.1.2 e pela Equação 4.1 é definido a seguinte equação para a *transformada de distância* [Ser82, SM92]:

$$\Psi_d(f) = \varepsilon_{b_k}(\cdots(\varepsilon_{b_1}(f))\cdots), \tag{4.4}$$

onde $b_G = b_1 \oplus \cdots \oplus b_k$ e o valor de k deve ser o maior valor de distância que pode ocorrer na imagem f. Como ilustração, veja Figura 4.3.

Se for aplicado k vezes o Algoritmo 4 da erosão, apresentado na Subseção 3.1.1, então a TD é computada. Além disso, a mesma saída é encontrada se for aplicado recursivamente este algoritmo até a estabilização. Este segundo caso tem como resultado a TD por causa da propriedade de operação idempotente, Equação 4.3, e é apresentado no Algoritmo 10.

Outros tipos de funções estruturantes aplicáveis a Equação 4.4 estão

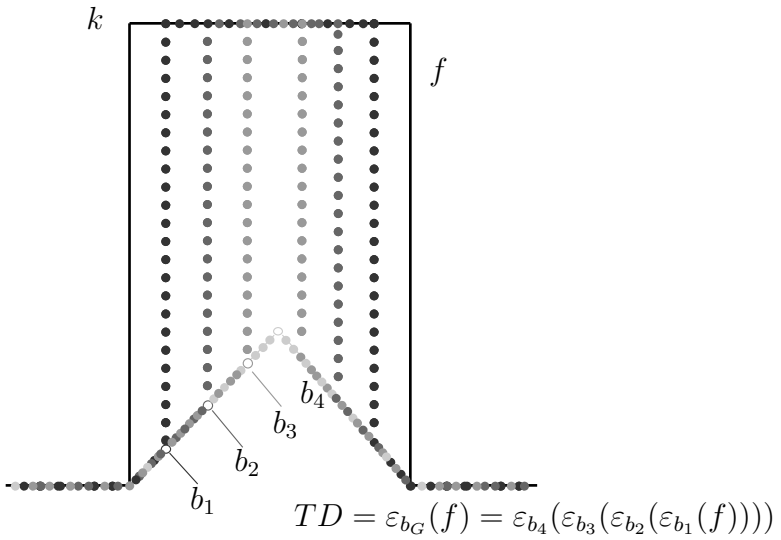


Figura 4.3: TD usando erosões por decomposições de funções estruturantes unidimensionais.

ilustrados nas Figuras 4.4a, 4.4b e 4.4c. Estes exemplos mostram funções estruturantes $b_i = b$, onde $b_G = kb$ para as métricas *city-block*, *chessboard* e *octagonal*, respectivamente. Estas métricas estão ilustradas também nas Figuras 4.7a, 4.7b e 4.7c, respectivamente.

$$\begin{array}{ccc} & -1 & \\ -1 & \mathbf{0} & -1 \\ & -1 & \end{array} \qquad \begin{array}{ccc} -1 & -1 & -1 \\ -1 & \mathbf{0} & -1 \\ -1 & -1 & -1 \end{array} \qquad \begin{array}{ccccc} & & -2 & & \\ & -2 & -1 & -2 & \\ -2 & -1 & \mathbf{0} & -1 & -2 \\ & -2 & -1 & -2 & \\ & & -2 & & \end{array}$$

(a) b_4 (b) b_8 (c) $b_O = b_4 \oplus b_8$

Figura 4.4: Funções estruturantes usadas nas decomposições das métricas (a) *city-block*, (b) *chessboard* e (c) *octagonal*, onde as origens estão em negrito.

Outros tipos de decomposições para b_G são as métricas *chanfrada 3:4* (Figuras 4.5a e 4.7d) e *chanfrada 5:7:11* (Figuras 4.5b e 4.7e). O trabalho de Borgefors [Bor86] trouxe um estudo completo da aproximação destas e

de outras métricas com a métrica *euclidiana*. Por exemplo, se ao invés de usar $a = 3$ e $b = 4$ na métrica *chanfrada 3:4* usar $a = 2$ e $b = 3$ (ou seja, *chanfrada 2:3*) a diferença com a métrica euclidiana vai de 0.0809 para 0.1340 aplicada em uma mesma imagem de um certo tamanho [Bor86]. Os algoritmos apresentados por Borgefors são semelhantes aos de Rosenfeld e Pfaltz [RP66, RP68] e poderão ser reescritos por erosão, como apresentados a seguir.

$$\begin{array}{ccc} -4 & -3 & -4 \\ -3 & \mathbf{0} & -3 \\ -4 & -3 & -4 \end{array}$$

(a) $b_{3:4}$

$$\begin{array}{ccccc} & & -11 & & -11 & \\ & -11 & -7 & -5 & -7 & -11 \\ & & -5 & \mathbf{0} & -5 & \\ & -11 & -7 & -5 & -7 & -11 \\ & & -11 & & -11 & \end{array}$$

(b) $b_{5:7:11}$

Figura 4.5: Funções estruturantes usadas nas decomposições das métricas (a) *chanfrada 3:4* e (b) *chanfrada 5:7:11*, onde as origens estão em negrito.

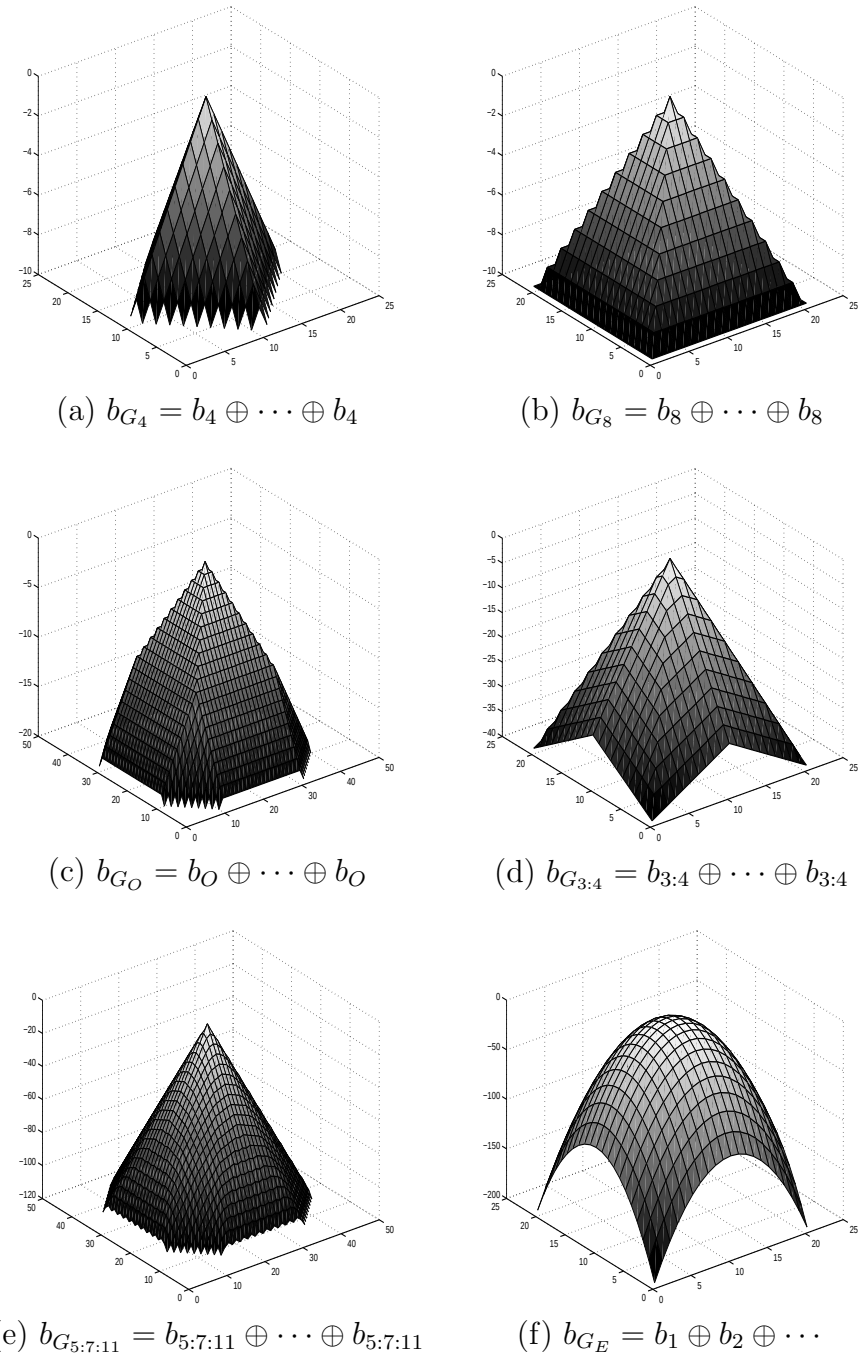
Huang e Mitchell [HM94] mostraram que a métrica euclidiana dada pela função estruturante b_{G_E} (Figura 4.7f) pode ser decomposta através de funções estruturantes 3×3 distintas, b_i, i inteiro positivo, veja Figura 4.6.

$-4i+2$	$-2i+1$	$-4i+2$
$-2i+1$	$\mathbf{0}$	$-2i+1$
$-4i+2$	$-2i+1$	$-4i+2$

Figura 4.6: b_i elemento estruturante usado na Equação 4.4 para obter a TDE, onde a origem está em negrito.

Reescrita de algoritmo paralelo da TD

Baseado na implementação da *erosão paralela*, Algoritmos 4 e 10, é possível reescrever os principais algoritmos da TD usando a teoria de ope-



(a) $b_{G_4} = b_4 \oplus \cdots \oplus b_4$ (b) $b_{G_8} = b_8 \oplus \cdots \oplus b_8$

(c) $b_{G_O} = b_O \oplus \cdots \oplus b_O$ (d) $b_{G_{3:4}} = b_{3:4} \oplus \cdots \oplus b_{3:4}$

(e) $b_{G_{5:7:11}} = b_{5:7:11} \oplus \cdots \oplus b_{5:7:11}$ (f) $b_{G_E} = b_1 \oplus b_2 \oplus \cdots$

Figura 4.7: Funções estruturantes obtidas através de 10 somas de Minkowski de suas respectivas decomposições.

Algoritmo 10 TD paralela

$TD : K^{\mathbb{E}} \times \mathbb{Z}^B \times \mathbb{Z}^B \times \cdots \longrightarrow K^{\mathbb{E}}$
$\quad TD(f, b_1, b_2, \ldots) = g$

$g = f;$
$i = 0;$
$\forall g \neq \varepsilon_{b_i}(g)$
$\quad g = \varepsilon_{b_i}(g);$
$\quad i++;$

radores morfológicos caracterizados por decomposição de função estruturante.

O primeiro algoritmo paralelo para a TD foi definido por Rosenfeld e Pfaltz e publicado em 1968 [RP68]. Outro trabalho clássico que usa esta implementação foi definido por Borgefors [Bor86]. Em ambos os casos a imagem de entrada assume valores zeros e uns e são descritos abaixo:

$$v_{i,j}^m = \min_{(k,l)\in mask}\{v_{i+k,j+l}^{m-1} + c(k,l)\}, \text{ até estabilizar,} \tag{4.5}$$

onde $v_{i,j}^m$ é o valor do pixel na posição (i, j) da imagem na iteração m, (k, l) é a posição na máscara $mask$ e $c(k, l)$ é o valor da máscara em (k, l) $(c(0, 0) = 0)$.

Note que as Equações 2.5 e 3.4 são equivalentes a Equação 4.5 resultando na mesma TD, considerando $b = b_i = -mask$, $i = 1, \cdots, k$. Estas conclusões são descritas com mais detalhes no trabalho de Shih e Mitchell de 1992 [SM92].

A melhor análise destas duas equações considera $x = (i, j)$ e $y = (i+k, j+l)$. A Equação 4.5 pode ser definida como $v^m(x) = \min\{v^{m-1}(y) \dot{-} b(y - x) : y \in B_x\}$, onde $x \in \mathbb{E}$. Note que, se $v^0 = f$ e $v^1 = \varepsilon_b(f)$, então é encontrado a definição da erosão da Equação 2.5. Como foi visto, para calcular a TD usando esta erosão é preciso considerar uma imagem com valores zeros e k (maior distância possível na imagem) e computar a erosão até a estabilização.

TDE em paralelo

Como exemplo de TDE usando uma *erosão paralela*, a Figura 4.8 mostra a função estruturante b_{G_E}, onde a origem está no topo do parabolóide. Esta função estruturante pode ser usada para calcular a TDE na imagem f fazendo $\varepsilon_{b_{G_E}}(f)$.

Como foi visto na Subseção 3.1.2, a função estruturante euclidiana b_{G_E} pode ser escrita como a *soma de Minkowski generalizada em níveis de cinza* de diversas funções estruturantes $b_i \in \{b_1, \cdots, b_k\}$. Usando a formação dada pela Figura 4.6 é possível obter $b_{G_E} = b_1 \oplus \cdots \oplus b_k$ e $\varepsilon_{b_{G_E}}(f) = \varepsilon_{b_k}(\cdots(\varepsilon_{b_1}(f))\cdots)$ =TDE, usando o Algoritmo 10.

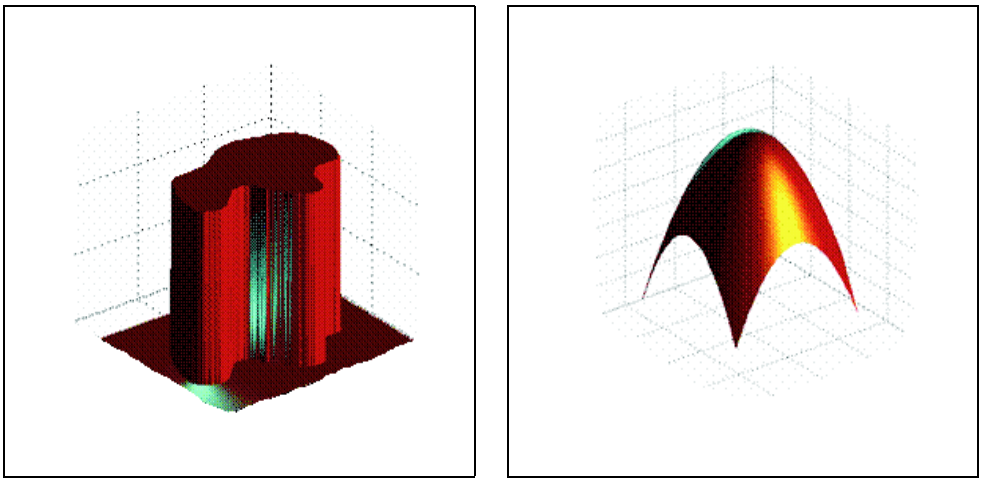
4.3.2 TD seqüencial

A erosão morfológica implementada de forma seqüencial foi vista na Seção 3.2. O algoritmo da TD seqüencial é exatamente o algoritmo da *erosão seqüencial*, Algoritmos 5 e 6, usando a decomposição $b_G = kb = kb^+ \oplus kb^-$. Veja um exemplo na Figura 4.9.

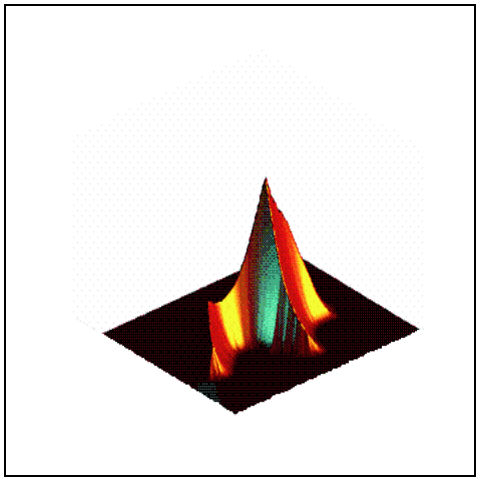
Note que não é possível usar uma decomposição com diferentes $b_i's$, como foram implementados nos padrões *paralelo* e *por propagação*. Portanto, não é possível calcular a métrica euclidiana usando apenas erosão seqüencial.

Reescrita de algoritmo seqüencial da TD

O primeiro algoritmo seqüencial da TD foi descrito por Rosenfeld e Pfaltz e publicado em 1966 [RP66]. Em seu algoritmo, a imagem de entrada contém somente 0/1, e o conjunto de pixels com valores zeros é não vazio. O algoritmo



f b_{G_E}



TDE

Figura 4.8: Imagem de entrada f, função estruturante euclidiana b_{G_E} e $TDE = \varepsilon_{b_{G_E}}(f)$.

-1	-1	-1
-1	**0**	-1
-1	-1	-1

b_8

-1	-1	-1
-1	**0**	

b_8^-

	0	-1
-1	-1	-1

b_8^+

k	k	k	k	k	k	k	k	k
k	k	k	k	k	k	k	k	k
k	k	k	k	k	k	k	k	k
k	k	k	k	0	k	k	k	k
k	k	k	k	k	k	k	k	k
k	k	k	k	k	k	k	k	k
k	k	k	k	k	k	k	k	k

(a) f

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	2	3	4
0	0	0	1	1	1	2	3	4
0	0	2	2	2	2	2	3	4
0	3	3	3	3	3	3	3	4

(b) $\varepsilon^+_{b_8^+}(f)$

4	3	3	3	3	3	3	3	4
4	3	2	2	2	2	2	3	4
4	3	2	1	1	1	2	3	4
4	3	2	1	0	1	2	3	4
4	3	2	1	1	1	2	3	4
4	3	2	2	2	2	2	3	4
4	3	3	3	3	3	3	3	4

(c) $\varepsilon^-_{b_8^-}(\varepsilon^+_{b_8^+}(f))$

Figura 4.9: (a) Imagem de entrada; (b) erosão *raster* por b_8^+ e (c) erosão *anti-raster* de (b) por b_8^-.

é dado por [2],

$$f_1(a_{i,j}) = \begin{cases} 0, \quad se\ a_{i,j} = 0, \\ \min(a_{i-1,j} + 1, a_{i,j-1} + 1), \quad se\ (i,j) \neq (1,1)\ e\ a_{i,j} = 1, \\ m + n, \quad se\ (i,j) = (1,1)\ e\ a_{i,j} = 1,\ e \end{cases} \tag{4.6}$$

$$f_2(a_{i,j}) = \min(a_{i,j}, a_{i+1,j} + 1, a_{i,j+1} + 1),$$

onde $a_{i,j}$ é o valor do pixel na posição (i, j) em uma imagem com m colunas e n linhas. Os valores $a_{i,j}$ fora da imagem não são definidos. f_1 é aplicada na ordem *raster* e, sobre o resultado, f_2 é aplicada na ordem *anti-raster*. Esta TD usa a métrica *city-block*. Se $a_{1,1} = 1$, o algoritmo faz $f_1(a_{1,1}) = m + n$, que é a maior distância possível na imagem. Este passo no algoritmo pode ser eliminado se for assumido uma imagem de entrada com valores zeros e k (k é a maior distância possível na imagem). Além disso, a primeira linha também pode ser eliminada se for incluída na função *min* três argumentos em vez de dois. Assim, $f_1(a_{i,j}) = min(a_{i,j}, a_{i-1,j} + 1, a_{i,j-1} + 1)$ pode ser substituída no passo f_1 do algoritmo acima.

Fazendo b^+ e b^- funções que decompõem a métrica *city-block* para as varreduras *raster* e *anti-raster*, respectivamente, como mostra a Figura 4.10, é possível reescrever a Equação 4.6 como:

$$f_1(a_{i,j}) = min\{a_{i,j} - b^+_{0,0}, a_{i+1,j} - b^+_{1,0}, a_{i,j+1} - b^+_{0,1}\},$$
$$f_2(a_{i,j}) = min\{a_{i,j} - b^-_{0,0}, a_{i-1,j} - b^-_{-1,0}, a_{i,j-1} - b^-_{0,-1}\}.$$

Se $x = (i, j)$ e $f(x) = a_{i,j}$, então $f_1(a_{i,j}) = \varepsilon^+_{b^+}(f)(x)$ e $f_2(a_{i,j}) = \varepsilon^-_{b^-}(f)(x)$, como definidos nas Equações 3.5 e 3.6, respectivamente. O algoritmo da TD de Rosenfeld e Pfaltz [RP66] é então equivalente a nossa *erosão*

[2] Observe que neste algoritmo o domínio da imagem está nos intervalos $1 \leq i \leq m$ e $1 \leq j \leq n$, diferente do definido na Seção 3.2.1, porém isto não afeta a nossa análise.

$$\begin{array}{cc} & -1 \\ -1 & \mathbf{0} \end{array} \qquad \begin{array}{cc} \mathbf{0} & -1 \\ -1 & \end{array}$$

$$b_4^- \qquad\qquad b_4^+$$

Figura 4.10: Decomposição *anti-raster* b_4^- e *raster* b_4^+ para a métrica *city-block*, com valor zero em negrito no centro.

seqüencial, Algoritmos 5 e 6. Borgefors [Bor86] também implementou um algoritmo seqüencial para a TD, o qual é também proposto aqui.

4.3.3 TD por propagação

Serão apresentados a seguir algoritmos da TD que usam a *erosão por propagação*, Algoritmo 8, e a *erosão por propagação generalizada*, Algoritmo 9.

Reescrita de algoritmo por propagação da TD

Vincent [Vin92] implementou a TD por propagação a qual está apresentada no Algoritmo 11. Este algoritmo usa uma estrutura de *fila* para armazenar a *fronteira*.

Nesse algoritmo, $FIFO\_add(x)$, $FIFO\_empty()$ e $FIFO\_first()$ são as primitivas de manipulação de fila que adiciona um elemento x na fila, verifica se a fila está vazia e retorna o primeiro elemento da fila, respectivamente.

Esse algoritmo pode ser reescrito como apresentado no Algoritmo 12, onde o primeiro $\forall$ coloca na fila todos os pixels de *fronteira* da imagem f. Este laço pode ser substituído pela função que encontra a fronteira, a qual coloca os pixels de fronteira no conjunto U como apresentado no código do Algoritmo 13. No laço $\forall\ FIFO\_empty() \neq \varnothing$ do Algoritmo 11, todos os

Algoritmo 11 TD por Vincent

$\mathrm{TD}^{Vinc} : \{0,1\}^{\mathbb{E}} \longrightarrow K^{\mathbb{E}}$
$\mathrm{TD}^{Vinc}(f) = f$ // trabalha na mesma imagem

$\forall x \in \mathbb{E}$ // B é a vizinhança 4 ou 8 definido pela grade quadrada
if $f(x) = 1$ and $\exists y \in B_x \cap \mathbb{E} \mid f(y) = 0$
$FIFO\_add(x)$;
$f(x) = 2$; // distância iniciada com valor 2 para a fronteira
$\forall\ FIFO\_empty() \neq \varnothing$
$x = FIFO\_first()$;
$\forall y \in B_x \cap \mathbb{E}$
if $f(y) = 1$
$f(y) = f(x) + 1$;
$FIFO\_add(y)$;

Algoritmo 12 TD auxiliar

$\mathrm{TD}^{aux} : \{0,k\}^{\mathbb{E}} \longrightarrow K^{\mathbb{E}}$
$\mathrm{TD}^{aux}(f) = f$ // $k = \infty$

$\forall x \in \mathbb{E}$
$\forall y \in B_x \cap \mathbb{E}$
if $f(y) > f(x) + 1$
$FIFO\_add(x)$;
$f(y) = 1$;
$\forall\ FIFO\_empty() \neq \varnothing$
$x = FIFO\_first()$;
$\forall y \in B_x \cap \mathbb{E}$
if $f(y) > f(x) + 1$
$f(y) = f(x) + 1$;
$FIFO\_add(y)$;

pixels de fronteira (da *fila*) são usados para computar a erosão por seus pixels vizinhos e ao mesmo tempo um novo pixel de fronteira é computado para verificar se será inserido na fila. Este processo é um caso particular do Algoritmo 8 - *Erosão por Propagação*, apresentado anteriormente, pois considera uma métrica fixa (*city-block* ou *chessboard*) em vez de usar uma função estruturante genérica b. Observe que não é necessário usar filas, mas dois conjuntos, que são trocados em cada iteração.

No Algoritmo 13 é apresentado uma generalização do algoritmo proposto por Vincent [Vin92].

Algoritmo 13 TD por propagação

$TD^p : K^{\mathbb{E}} \times \mathbb{Z}^B \longrightarrow K^{\mathbb{E}}$
$TD^p(f, b) = g$

$g = \varepsilon_b(f)$;
$U = \partial(g, b)$;
$\forall\ U \neq \emptyset$ // até estabilizar
$(g, U) = \varepsilon^p(g, b, U)$;

TDE por propagação

O algoritmo da TDE por propagação é o Algoritmo 9 - *Erosão por propagação generalizada*, usando as decomposições de $b_{G_E} = b_1 \oplus \cdots \oplus b_k$, onde os $b_i's$ foram definidos na Figura 4.6 e b_{G_E} é ilustrado na Figura 4.7f. Este algoritmo, bastante simples, apresenta uma boa eficiência quando implementado em máquinas seqüenciais de propósito geral, como analisado na Seção 4.7 e resumido na Tabela 4.2.

4.4 TD por propagação direcional

Ragnemalm [Rag92] implementou a TDE utilizando duas estruturas de filas para armazenar os pixels de *fronteira* e várias condições de propagação, onde para cada direção de propagação existe um teste, gerando um algoritmo complexo. Eggers [Egg98] melhorou o algoritmo de Ragnemalm diminuindo os vários testes de direção, porém criando inúmeras variáveis, deixando o código mais eficiente, contudo não menos complexo. A idéia proposta por Eggers é fazer uso da direção da propagação para restringir a varredura da vizinhança apenas aos pixels que podem ser modificados. Inspirado neste princípio, é possível alterar o Algoritmo 9 - *Erosão por propagação generalizada*, para que as erosões também façam uso da direção da propagação, como será apresentado a seguir.

Considere uma outra versão para o cômputo da fronteira ∂, onde além de armazenar a coordenada x, armazena também a direção de propagação da erosão definida por um inteiro $k \in \{1, \ldots, 8\}$, cuja direção corresponde a propagação de x, conforme a Figura 4.11.

5	6	7
4		8
3	2	1

Figura 4.11: Direções de propagação.

Alterando o Algoritmo 9 - *Erosão por propagação generalizada*, para fazer uso da direção de propagação, é apresentado o Algoritmo 14 (agora U contém $(x, d) \in \mathbb{E}'$, onde $\mathbb{E}' = \mathbb{E} \times \{1, \ldots, 8\}$, x é a coordenada e d é a direção de propagação).

Os comandos ε^{init} e ε^{dir} que aparecem no Algoritmo 14 estão definidos nos Algoritmos 15 e 16, respectivamente.

No Algoritmo 16, $B'_x[k{-}{-}, k, k{+}{+}]$ é o subconjunto das direções: *anterior*,

Algoritmo 14 TD por propagação direcional

$\mathrm{TD}^{dir} : K^{\mathbb{E}} \times \mathbb{Z}^B \times \mathbb{Z}^B \times \cdots \longrightarrow K^{\mathbb{E}}$
$\mathrm{TD}^{dir}(f, b_1, b_2, \cdots) = g$

$[g, U] = \varepsilon^{init}(f, b_1)$;
$i = 2$;
$\forall U \neq \emptyset$
  $(g, U) = \varepsilon^{dir}(f, b_i, U)$;
  $i{+}{+}$;

Algoritmo 15 Erosão inicial

$\varepsilon^{init} : K^{\mathbb{E}} \times \mathbb{Z}^B \longrightarrow K^{\mathbb{E}} \times \mathbb{E}'$ // $\mathbb{E}' = \mathbb{E} \times \{1, \ldots, 8\}$
$\varepsilon^{init}(f, b) = (g, U)$

$g = f$;
$U = \emptyset$;
$\forall x \in \mathbb{E}$
  $\forall y \in B_x \cap \mathbb{E}$
    if $g(x) > f(y) \dot{-} b(y - x)$
      $g(x) = f(y) \dot{-} b(y - x)$;
      if $x \notin U$
        $U = U \cup \{(x, d)\}$; // d é a direção de B

k e *posterior* a k, considerando o sentido horário, como ilustrado na Figura 4.11. A direção k é a direção de propagação de x obtida numa iteração anterior. Se $k = 1$, então *anterior* é 8, e se $k = 8$, *posterior* é 1. Na última linha do código, d é $k{-}{-}$, k ou $k{+}{+}$ dependendo da direção do vetor com origem x e destino y.

Algoritmo 16 Erosão por propagação direcional

$\varepsilon^{dir} : K^{\mathbb{E}} \times \mathbb{Z}^B \times \mathbb{E}' \longrightarrow K^{\mathbb{E}} \times \mathbb{E}' \quad // \;\; \mathbb{E}' = \mathbb{E} \times \{1, \ldots, 8\}$
$\varepsilon^{dir}(f, b, U) = (g, U')$

$g = f;$
$\forall\ (x, k) \in U$
  $\forall y \in B'_x[k{-}{-}, k, k{+}{+}] \cap \mathbb{E}$
    if $g(y) > f(x) \dot{-} b(x - y)$
      $g(y) = f(x) \dot{-} b(x - y);$
      $U = U \cup \{(y, d)\};$ // d é a direção de B

Se forem usadas as funções estruturantes $b'_i s$, como definidas na Figura 4.6, o Algoritmo 14 encontra a TDE, que é similar em vários aspectos ao proposto por Eggers [Egg98], porém mais simples e com desempenhos equivalentes, como analisados na Seção 4.7.

A Figura 4.12 ilustra uma iteração intermediária da TD por propagação direcional da imagem de entrada f pelas funções estruturantes $b_1, b_2, \ldots$ definidas pela Figura 4.6. A função estruturante que está sendo processado na iteração é b_2 com a direção de propagação 4, o que implica ao algoritmo processar apenas as direções $[3, 4, 5]$, conforme Figura 4.11. O pixel x, que está sendo processado, é o em negrito e está recebendo o valor $\mathbf{4} = \bigwedge\{1\dot{-}(-6), 1\dot{-}(-3), 1\dot{-}(-6)\} = \mathrm{TD}^{dir}(f, b_1, b_2, \ldots)(x)$.

$$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & k & k & k & k & k \\ 0 & 0 & k & k & k & k & k \\ 0 & 0 & k & k & k & k & k \\ 0 & 0 & k & k & k & k & k \\ 0 & 0 & k & k & k & k & k \end{array}$$

f

$$\begin{array}{ccc} -2 & -1 & -2 \\ -1 & \mathbf{0} & -1 \\ -2 & -1 & -2 \end{array}$$

b_1

$$\begin{array}{ccc} -6 & -3 & -6 \\ -3 & \mathbf{0} & -3 \\ -6 & -3 & -6 \end{array}$$

b_2

$$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 & 4 & 4 & 4 \\ 0 & 0 & 1 & \mathbf{4} & k & k & k \\ 0 & 0 & 1 & 4 & k & k & k \\ 0 & 0 & 1 & 4 & k & k & k \end{array}$$

$\mathrm{TD}^{dir}(f, b_1, b_2, \ldots)(x)$

Figura 4.12: Ilustração de uma iteração intermediária da TD por propagação direcional, $\mathrm{TD}^{dir}(f, b_1, b_2, \ldots)(x)$, da imagem de entrada f pelas funções estruturantes $b_1, b_2, \ldots$, onde o pixel x, com valor **4** em negrito, é o que está sendo processado e a direção propagada é 4, conforme Figura 4.11.

4.5 TDE multidimensional

Será apresentado nesta seção um algoritmo da TDE multidimensional formulado usando erosões morfológicas. A TD é composta de várias erosões unidimensionais (1D). As funções estruturantes usadas na erosão pertencem a uma família de quatro funções estruturantes direcionais formadas por dois pixels. Duas para erosões seqüencias e outras duas para erosões por propagação.

Como foi visto na Subseção 3.1.2, a função estruturante euclidiana b_{G_E} pode ser escrita como a soma de Minkowski em níveis de cinza de diversas funções estruturantes $b_i's \in \{b_1, \cdots, b_k\}$. Usando a formação dada pela Figura 4.6 é possível obter $b_{G_E} = b_1 \oplus \cdots \oplus b_k$ e $\varepsilon_{b_{G_E}}(f) = \varepsilon_{b_k}(\cdots(\varepsilon_{b_1}(f))\cdots)$.

Note que esses $b_i's$ podem ainda ser decompostos em quatro funções estruturantes direcionais de dois pixels, duas nas direções verticais, Norte (b_{Ni}), e Sul (b_{Si}), e duas nas direções horizontais, Leste (b_{Ei}), e Oeste (b_{Wi}):

$$b_{Ni} = \begin{bmatrix} -2i+1 \\ \mathbf{0} \end{bmatrix},$$

$$b_{Wi} = \begin{bmatrix} -2i+1 & \mathbf{0} \end{bmatrix}, \qquad b_{Ei} = \begin{bmatrix} \mathbf{0} & -2i+1 \end{bmatrix},$$

$$b_{Si} = \begin{bmatrix} \mathbf{0} \\ -2i+1 \end{bmatrix}. \tag{4.7}$$

A função estruturante para a TDE pode ser decomposta em:

$$\begin{aligned} b_E \;=\; & \cdots \oplus b_{N2} \oplus b_{N1} \oplus \cdots \oplus b_{S2} \oplus b_{S1} \oplus \cdots \\ & \cdots \oplus b_{W2} \oplus b_{W1} \oplus \cdots \oplus b_{E2} \oplus b_{E1}. \end{aligned} \tag{4.8}$$

Esta decomposição gera duas conseqüências importantes. Primeiro, a

TDE pode ser separada em mais erosões 1D simples, que traz independência para a computação de cada linha ou coluna da imagem. Segundo, as funções estruturantes são reduzidas para funções estruturantes direcionais de dois pixels, permitindo implementações simples e eficiente de algoritmos.

Isto significa que a TDE pode ser computada pela erosão de cada coluna da imagem por $b_{N1}, b_{N2}, \ldots$ até a estabilidade usando a propriedade idempotente, então pela erosão das colunas por $b_{S1}, b_{S2}, \ldots$, seguida pela erosão das linhas por $b_{E1}, b_{E2}, \ldots$, e finalmente, pela erosão das linhas por $b_{W1}, b_{W2}, \ldots$.

4.5.1 TDE unidimensional

Nos algoritmos paralelos, os pixels são processados independentemente da ordem de varredura. Os pixels alterados na transformação dependem apenas dos pixels da imagem de entrada e da função estruturante, como foi visto no Algoritmo 4 - *Erosão paralela*.

O Algoritmo 17 da TDE 1D apresentado a seguir é exato e um dos mais simples para o cômputo da distância euclidiana encontrados na literatura.

A ineficiência deste algoritmo ocorre devido à varredura desnecessária nas áreas da imagem onde a erosão não é afetada. A melhor eficiência pode ser verificada com algoritmos por propagação.

4.5.2 TDE por propagação unidimensional

A idéia do algoritmo de *erosão por propagação* está em processar somente os pixels que podem mudar na erosão. Este conjunto de pixels é chamado *fronteira* de f, denotado por $\partial(f, b)$. Neste caso, a fronteira para a próxima erosão é computada durante a erosão prévia. Como foi visto nos Algoritmos 8 e 13, da erosão por propagação e da TD por propagação, respectivamente.

Para as erosões usando a família de funções estruturantes decomposta

Algoritmo 17 TDE 1D clássica

$\text{TDE}^{1D} : K^{\mathbb{E}} \longrightarrow K^{\mathbb{E}}$
$\quad \text{TDE}^{1D}(f) = g$
// f é imagem binária de entrada com valores 0 e M,
// onde M é o quadrado da distância máxima na imagem

$\forall\ c \subset \mathbb{E}$ // primeiro passo, erosões verticais da coluna c
$\quad \forall\ b = 1, 3, 5, 7, \ldots$ // até estabilizar
$\quad\quad \forall\ r \subset \mathbb{E}$ // varre a coluna c
$\quad\quad\quad N(r, c) = \min(f(r, c), f(r - 1, c) + b)$
$\quad \forall\ b = 1, 3, 5, 7, \ldots$ // até estabilizar
$\quad\quad \forall\ r \subset \mathbb{E}$
$\quad\quad\quad S(r, c) = \min(N(r, c), N(r + 1, c) + b)$
$\forall\ r \subset \mathbb{E}$ // segundo passo, erosões horizontais da linha r
$\quad \forall\ b = 1, 3, 5, 7, \ldots$ // até estabilizar
$\quad\quad \forall\ c \subset \mathbb{E}$
$\quad\quad\quad E(r, c) = \min(S(r, c), S(r, c + 1) + b)$
$\quad \forall\ b = 1, 3, 5, 7, \ldots$ // até estabilizar
$\quad\quad \forall\ c \subset \mathbb{E}$
$\quad\quad\quad g(r, c) = \min(E(r, c), E(r, c - 1) + b)$

// Considere $f(r - 1, c), N(r + 1, c), S(r, c + 1)$ e $E(r, c - 1)$ tratados
// na função min com valor M fora do domínio $\mathbb{E}$.

em dois pixels, é possível melhorar a eficiência do algoritmo por propagação. Estas melhorias são diferentes para cada um dos dois passos: a primeira *erosão seqüencial vertical* e a segunda *erosão por propagação horizontal*.

Parte 1

Para a primeira erosão seqüencial vertical será apresentado o Algoritmo 18[3].

Algoritmo 18 TDE 1D com erosão seqüencial vertical - parte 1

$\text{TDE}^{1D1} : K^{\mathbb{E}} \longrightarrow K^{\mathbb{E}}$
$\text{TDE}^{1D1}(f) = g$
// f é imagem de entrada e saída, H é altura da imagem

$\forall$ $c \subset \mathbb{E}$ // coluna
b=1;
$\forall$ $r = 2..H$ // varre a coluna c na ordem *raster*
if $f(r,c) > f(r-1,c) + b$
$f(r,c) = f(r-1,c) + b$;
$b = b + 2$;
else;
$b = 1$;
$b = 1$;
$\forall$ $r = (H-1)..1$ // varre a coluna c na ordem *anti-raster*
if $f(r,c) > f(r+1,c) + b$
$f(r,c) = f(r+1,c) + b$;
$b = b + 2$;
else
$b = 1$;

Como a imagem de entrada é binária, a fronteira pode ser propagada na

[3]O Algoritmo 18 pode ser melhorado da seguinte forma: na varredura *raster*, quando $f(r,c) > f(r-1,c) + b$ for verdade, inicia-se um contador *count* (inicializado com 0) que finaliza quando a desigualdade assumir falso. A partir deste ponto inicia-se uma varredura *anti-raster* até *count*/2 utilizando o segundo **if** do algoritmo. Este processo se repete para todos os objetos e todas as colunas e é ilustrado na Figura 4.16.

mesma direção da função estruturante usando um processamento seqüencial de forma que o pixel modificado na transformação é uma função do pixel modificado previamente. Deste modo a erosão vertical pode ser computada usando uma única varredura *raster* e uma única varredura *anti-raster*.

Parte 2

Para a segunda parte, erosão por propagação horizontal, será apresentado o Algoritmo 19.

Nessa erosão horizontal, não é possível fazer um processamento seqüencial, mas é possível evitar a cópia das imagens e processar a imagem desde que a ordem de processamento seja a ordem oposta da propagação da fronteira. Quando se usa a direção Leste, a ordem *raster* deve ser Oeste, de forma que o pixel modificado não será usado naquela varredura. Isto é alcançado usando *fila FIFO - First-In-First-Out*, duas para cada direção. A eficiência deste algoritmo é melhorada pelo fato de que só os pixels de fronteira são processados até a estabilidade.

Note que as filas usadas neste algoritmo têm um tamanho de máximo na largura da imagem. Para melhor desempenho, estas filas podem ser facilmente implementadas por vetores de inteiro de comprimentos fixos.

Observe que nos Algoritmos 18 e 19 de TDE as erosão não estão definidas como funções separadas, como foi feito em todos os algoritmos até este ponto, pois o principal objetivo destes algoritmos é mostrar uma implementação eficiente. Estas duas versões estão no trabalho de Lotufo e Zampirolli [LZ01]. Contudo, no Algoritmo 18, o leitor pode ver que o laço $\forall \;\; r = 2..H$ faz a *erosão seqüencial* da coluna c na ordem *raster*, enquanto o laço $\forall \;\; r = (H-1)..1$ faz a *erosão seqüencial* da mesma coluna c na ordem *anti-raster*. O mesmo ocorre no Algoritmo 19, onde o laço **while** $notEmptyQueue(Eq)$ faz a erosão por propagação para a direita e **while** $notEmptyQueue(Wq)$ faz a *erosão por propagação* para a esquerda, até que as estruturas Eq e Wq fiquem vazias.

Algoritmo 19 TDE 1D com erosão por propagação horizontal - parte 2

$\text{TDE}^{1D2} : K^{\mathbb{E}} \longrightarrow K^{\mathbb{E}}$
$\text{TDE}^{1D2}(f) = g$
// f é imagem de entrada e saída, W é a largura da imagem

```
∀ r ⊂ 𝔼, // linha
    ∀ c = (W − 1)..1
        insertQueue(Eq, c);
    ∀ c = 2..W
        insertQueue(Wq, c);
    b = 1;
    while (notEmptyQueue(Wq) ou notEmptyQueue(Eq))
        while notEmptyQueue(Eq)
            c = fromQueue(Eq)
            if f(r, c + 1) > f(r, c) + b
                f(r, c + 1) = f(r, c) + b;
                if c + 1 < W
                    insertQueue(Eq2, c + 1);
        while notEmptyQueue(Wq)
            c = fromQueue(Wq);
            if f(r, c − 1) > f(r, c) + b
                f(r, c − 1) = f(r, c) + b;
                if c − 1 > 1
                    insertQueue(Wq2, c − 1);
        b = b + 2;
        Wq = Wq2;
        Eq = Eq2;
```

Para melhor ilustrar o algoritmo da TDE multidimensional usando erosões serão apresentados a seguir alguns exemplos.

Exemplo 1

Neste primeiro exemplo é considerado apenas uma transformação 1D para melhor entendimento da erosão morfológica usada para computar a métrica euclidiana. A Figura 4.13 ilustra a erosão de f pela função estruturante b_{G_E}. A Figura 4.14 ilustrada a soma de Minkowski de b_1 por b_2. A generalização forma $b_{G_E} = b_1 \oplus \cdots \oplus b_k$. A Figura 4.15 ilustra a TDE obtida pela composição de erosões.

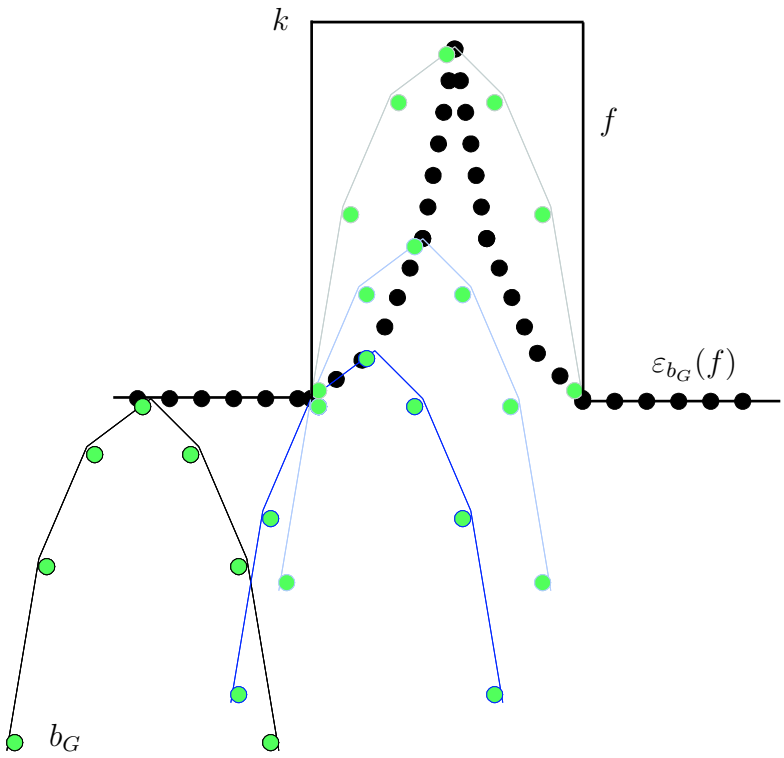


Figura 4.13: Ilustração da TDE 1D obtida da erosão pela função estruturante b_{G_E}.

$$b_i = \boxed{-2i+1} \boxed{\mathbf{0}} \boxed{-2i+1}$$

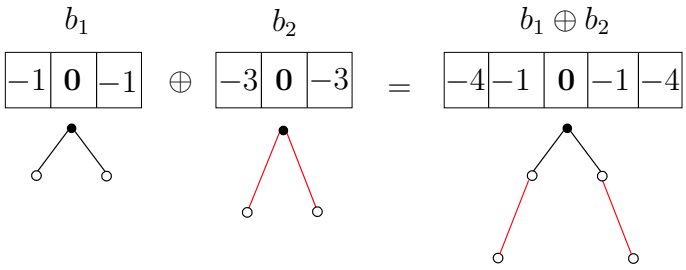


Figura 4.14: Soma de Minkowski em níveis de cinza para a métrica euclidiana, onde as origens estão no centro de cada função estruturante.

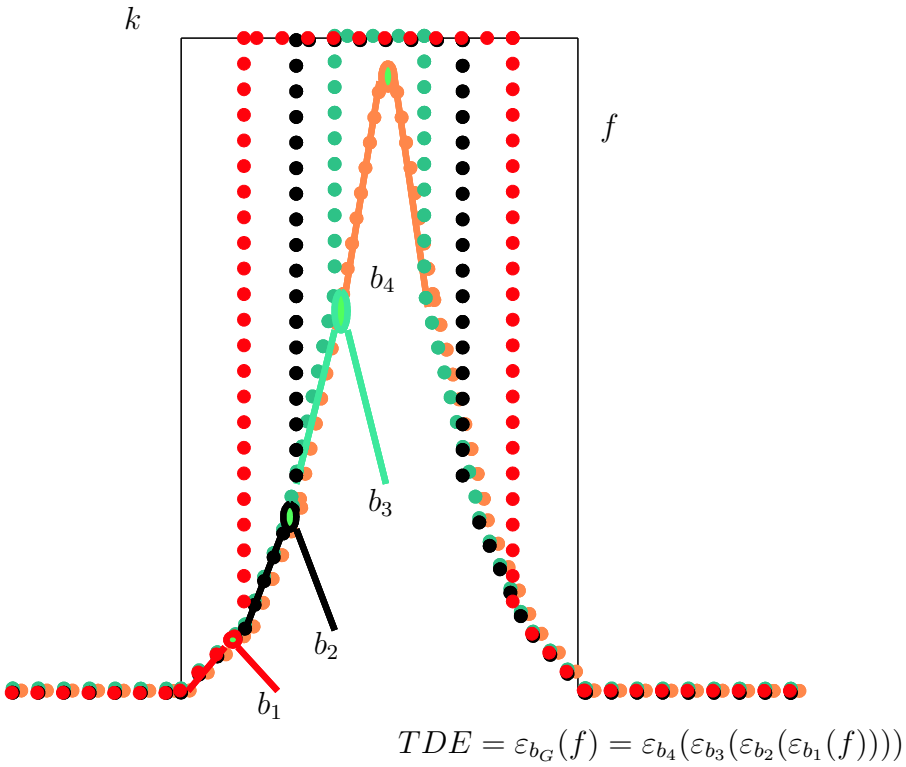
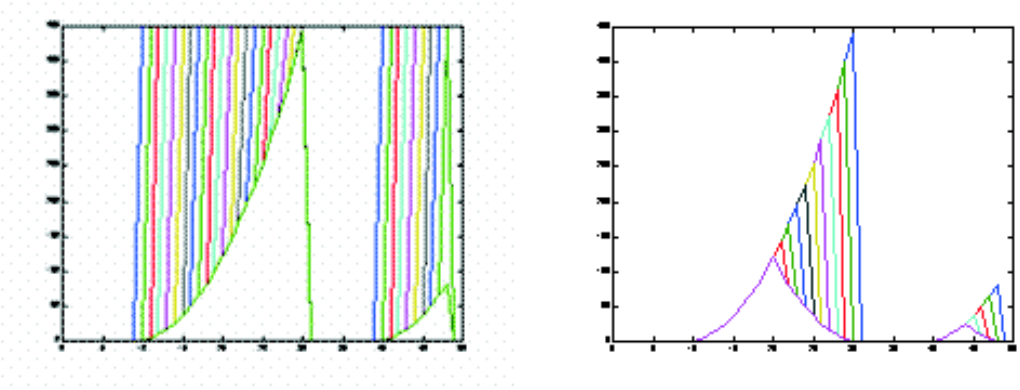


Figura 4.15: Ilustração da TDE 1D obtida de erosões por decomposição de função estruturante.

Exemplo 2

Para visualizar as transformações ocorridas nas quatro erosões do algoritmo multidimensional, a Figura 4.16 ilustra o primeiro passo do algoritmo, onde são computadas as erosões seqüenciais *raster* e *anti-raster*. Enquanto que a Figura 4.17 ilustra do segundo passo deste algoritmo, onde são realizadas as erosões por propagação.



raster *anti-raster*

Figura 4.16: Ilustração do primeiro passo do algoritmo multidimensional.

Exemplo 3

Finalmente, um exemplo simples usando uma imagem f de tamanho 4×4 é mostrado na Figura 4.18. O resultado da erosão vertical pela função estruturante B_v é apresentado na função f_v. Em seguida, f_1 é a primeira erosão por propagação horizontal e f_2 é a segunda erosão por propagação horizontal representando o quadrado da distância euclidiana. Nos resultados das erosões horizontais, os pixels inseridos na fila de propagação são marcados em negrito.

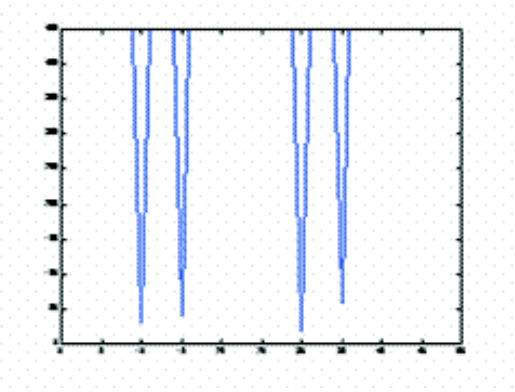
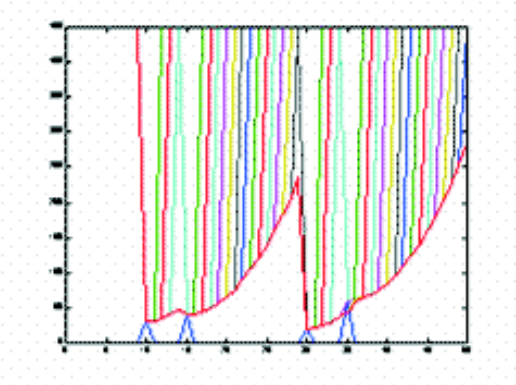
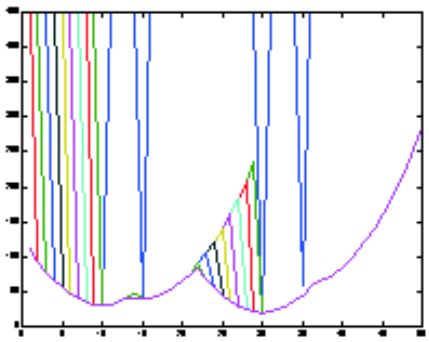


Imagem de Entrada



East



West

Figura 4.17: Ilustração do segundo passo do algoritmo multidimensional.

$$f = \begin{pmatrix} \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ 0 & \infty & \infty & \infty \end{pmatrix}, \qquad B_v = \begin{pmatrix} -5 \\ -3 \\ -1 \\ 0 \\ -1 \\ -3 \\ -5 \end{pmatrix},$$

$$f_v = f \ominus B_v = \begin{pmatrix} 9 & 4 & 0 & \infty \\ 4 & 1 & 1 & \infty \\ 1 & 0 & 4 & \infty \\ 0 & 1 & 9 & \infty \end{pmatrix},$$

$$B_{h1} = \begin{pmatrix} \text{-1} & 0 & \text{-1} \end{pmatrix}, \qquad f_1 = f_v \ominus B_{h1} = \begin{pmatrix} \mathbf{5} & \mathbf{1} & 0 & \mathbf{1} \\ \mathbf{2} & 1 & 1 & \mathbf{2} \\ 1 & 0 & \mathbf{1} & \mathbf{5} \\ 0 & 1 & \mathbf{2} & \mathbf{10} \end{pmatrix},$$

$$B_{h2} = \begin{pmatrix} -3 & 0 & -3 \end{pmatrix}, \qquad f_2 = f_1 \ominus B_{h2} = \begin{pmatrix} \mathbf{4} & 1 & 0 & 1 \\ 2 & 1 & 1 & 2 \\ 1 & 0 & 1 & \mathbf{4} \\ 0 & 1 & 2 & \mathbf{5} \end{pmatrix}.$$

Figura 4.18: Aplicação da TDE multidimensional.

	Algoritmos			**Métricas**					
DT	*Paralelo*	*Seqüencial*	*Propagação*	b_4	b_8	b_O	$b_{3:4}$	$b_{5:7:11}$	b_E
[RP66]		X		X					
[RP68]	X			X	X	X			
[Bor86]	X	X		X	X	X	X	X	X
[SM92]	X			X	X		X	X	X
[Vin92]	X	X	X	X	X	X			
[HM94]	X								X

Tabela 4.1: Classificação de algoritmos da TD e de suas decomposições de funções estruturantes.

4.6 Resumo da classificação da TD usando erosões

A Tabela 4.1 resume a classificação dos principais algoritmos da TD nos padrões paralelo, seqüencial e por propagação e suas decomposições de funções estruturantes $b_4, b_8, b_O, b_{3:4}, b_{5:7:11}$ e b_E. Esta tabela também foi apresentada no artigo [ZL00].

Note que não foram incluídos nesta classificação os algoritmos para a TD por propagação direcional e a TDE multidimensional pois são casos particulares dos padrões apresentados neste texto.

4.7 Análise de desempenho e comparações

Serão apresentadas nesta seção as análises e as comparações dos algoritmos da TDE paralelo, por propagação, por propagação direcional e multidimensional vistos nas Seções 4.3, 4.4 e 4.5, respectivamente.

Comparando o algoritmo da TDE obtido por erosões direcionais, proposto na Seção 4.4, com o algoritmo do Eggers [Egg98] é possível notar que este segundo é mais eficiente (veja Tabela 4.2), isto porque não existem cópias de imagens nas iterações por conseqüência dos vários testes existes. A simplici-

	Imagem 1	**Imagem 2**
Saito	80	60
Eggers	100	90
PMN	90	80
Meijster	121	120
LZ	90	101
TDE^{Dir}	190	210
TDE^{Pro}	230	240
TDE^{Par}	1012	701

Tabela 4.2: Tempo em mili-segundos do desempenho de diversos algoritmos da TDE.

dade do algoritmo direcional proposto nesta tese elimina estes testes, porém para o cálculo da TDE é necessário fazer a cópia dos pixels de fronteira em cada iteração, que torna o nosso algoritmo ineficiente. Isto não ocorre quando se usa funções estruturantes constantes, como para as métricas *chanfradas*.

A Tabela 4.2 apresenta um resumo dos testes feitos para o cálculo da TDE de diversos algoritmos da literatura e também dos algoritmos apresentados nesta tese. Para estes testes foi usado um *Pentium III* $-$ 800 MHz $-$ 256 MB. A **Imagem 1** e a **Imagem 2** são imagens 512×512 ilustradas na Figura 4.22 (a) e (b), respectivamente. As implementações dos três primeiros algoritmos apresentados na tabela (Saito [ST94], Eggers [Egg98] e PMN [Cui99]) foram extraídas da *home page* do Olivier Cuisenaire. O algoritmo Meijster é um dos mais recentes da literatura [MR00]. O algoritmo LZ é o multidimensional definido na Seção 4.5 e os restantes foram definidos nas Seções 4.3 e 4.4.

O bom desempenho dos algoritmos Saito [ST94] e PMN [Cui99] se deve ao fato de ambos usarem uma *lookup table* para armazenar o mapa da distância euclidiana [Cui99], veja Figura 4.19. Este recurso não foi incluído nos algoritmos propostos nesta tese, pois não iriam mais ser definidos explicitamete por erosões, que é a essência desta tese.

Meijster e Roerdink [MR00] definiu um dos algoritmos da TDE mais recentes, onde possui o primeiro passo igual ao algoritmo multidimensional

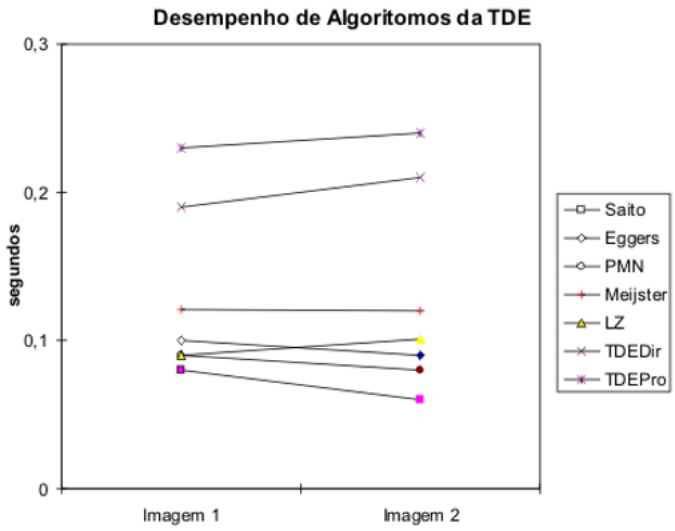


Figura 4.19: Gráfico ilustrando os desempenhos de algoritmos da TDE referente à Tabela 4.2.

proposto nesta tese. No segundo passo o algoritmo do Meijster e Roerdink usa uma multiplicação para cada pixel conferir a região dominante mais próxima. A vantagem do algoritmo multidimensional é que ele não exige multiplicação, só comparação e adição, que faz do algoritmo proposto um dos mais rápido na maioria das situações.

O primeiro passo do algoritmo multidimensional exige uma varredura *raster* e outra *anti-raster* com vizinhança de dois pixels cada. O desempenho do segundo passo é mais complexo. A melhor situação exige uma varredura *raster* e outra *anti-raster* com vizinhança de dois pixels, que acontece com uma imagem com colunas iguais, onde nenhuma propagação horizontal é exigida. Em um caso típico, a velocidade depende do número de vezes que o pixel entra na fila de propagação.

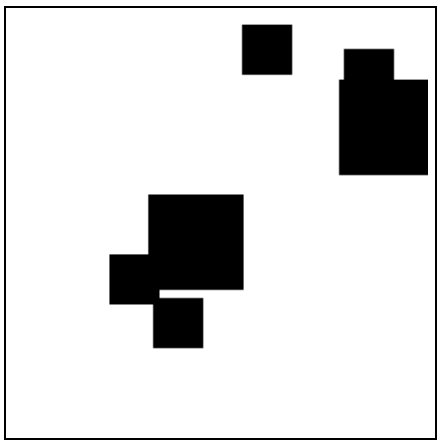
O algoritmo multidimensional proposto nesta tese tem uma patologia de pior caso que acontece com uma imagem quadrada com uma linha de fundo diagonal, veja Figura 4.22c. Nesta situação o número de vezes que um pixel é inserido na fila é aproximadamente $W/4$, onde W é a largura da imagem. Embora isto faça este algoritmo quadrático com relação às dimensões da imagem, isto dificilmente ocorre em imagens típicas. Uma análise mais precisa pode ser observada através da Figura 4.20, onde os valores mostram o número vezes que um pixel é inserido na fila no segundo passo do algoritmo multidimensional. Observe que no pior caso cada linha possui duas progressões aritméticas, assim o algoritmo possui complexidade $O(HW^2)$, onde W é a largura e H é a altura da imagem.

Para ilustrar o comportamento dos algoritmos com imagens de tamanhos diferentes, a imagem *box* (veja Figura 4.21) foi reproduzida por um fator de 4, 8, 16 e 64 e uma comparação de tempo foi feita usando a TDE definida por Meijster e Roerdink [MR00], veja Tabela 4.3.

Dos resultados apresentados nas Tabela 4.2 e 4.3, pode ser concluído que o algoritmo proposto tem um desempenho melhor que um dos mais rápidos da TDE e um desempenho comparável com algoritmos mais conhecidos da

4	4	3	3	2	2	1	1	0	0
4	3	3	2	2	1	1	0	0	0
3	3	2	2	1	1	0	0	0	1
3	2	2	1	1	0	0	0	1	1
2	2	1	1	0	0	0	1	1	2
2	1	1	0	0	0	1	1	2	2
1	1	0	0	0	1	1	2	2	3
1	0	0	0	1	1	2	2	3	3
0	0	0	1	1	2	2	3	3	4
0	0	1	1	2	2	3	3	4	4

Figura 4.20: Matriz representado o número de vezes que um pixel é inserido na fila no segundo passo do algoritmo multidimensional aplicado na imagem de pior caso (Figura 4.22c) de tamanho 10×10.



nbox

Figura 4.21: Imagem 256×256 utilizada para teste de eficiência para os algoritmos da TDE, onde a cor preta é 0 e a cor branca é $k \neq 0$.

	64K	256K	1M	4M
LZ	0.022	0.153	0.659	2.81
Meijster	0.028	0.181	0.763	3.08

Tabela 4.3: Tempo em segundos do algoritmo proposto e o do definido por Meijster e Roerdink [MR00] aplicados para a imagem *box* reproduzida por um fator de $4, 16, 32$ e 64.

distância euclidiana.

As experiências da Tabelas 4.3 foram feitas em um *notebook Pentium III,* $750MHz$*,* $128MB$*.*

Referências Bibliográficas

[BBL94] J. Barrera, G.F. Banon e R.A. Lotufo. A mathematical morphology toolbox for the KHOROS system. No *Image Algebra and Morphological Image Processing V*, páginas 241–252, Bellingham, Julho 1994. SPIE.

[Bor86] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986.

[Cui99] O. Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing.* Tese de Doutorado, Université Catholique de Louvain, Bélgica, 1999.

[Dan80] P.E. Danielsson. Euclidean distance mapping. *Computer Vision, Graphics and Image Processing*, 14:227–248, 1980.

[EBS01] E. Engbers, R.v.d. Boomgaard e A.W.M. Smeulders. Decomposition of separable concave structuring functions. *Journal of Mathematical Imaging and Vision*, 15(3), 2001.

[Egg98] H. Eggers. Two fast Euclidean distance transformations in $\mathbb{Z}^2$ based on sufficient propagation. *Computer Vision, Graphics and Image Processing*, 69(1), 1998.

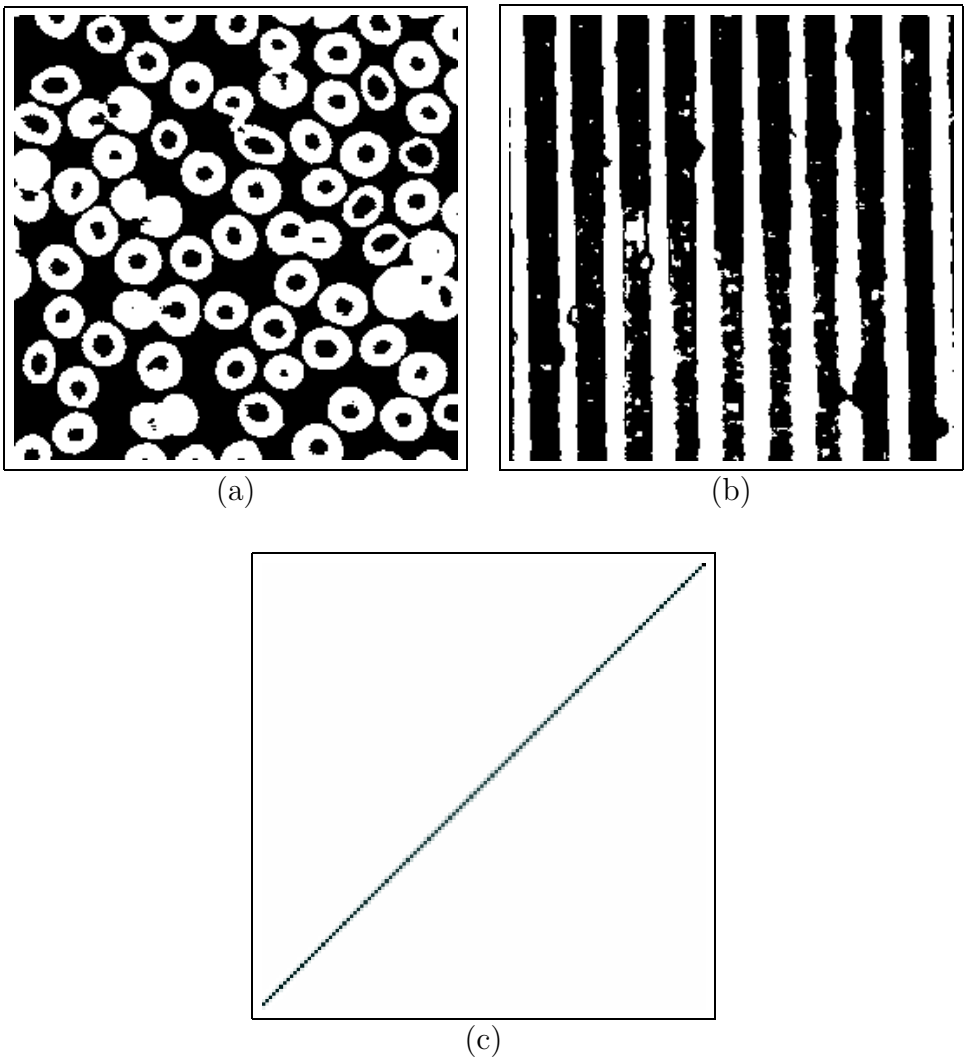


Figura 4.22: (a) e (b) imagens reais; (c) pior caso para a TDE multidimensional.

[HM94] C.T. Huang e O.R. Mitchell. A Euclidean distance transform using grayscale morphology decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:443–448, 1994.

[LFZ02] R.A. Lotufo, A.A. Falcão e F.A. Zampirolli. Ift-watershed from gray scale marker. No *Brazilian Symposium on Computer Graphics and Image Processing*, Fortaleza, RN, Brasil, Outubro 2002.

[LT00] N.J. Leite e M.D. Teixeira. An idempotent scale-space approach for morphological segmentation. No *Mathematical Morphology and its Applications to Image and Signal Processing*, volume 18, páginas 341–350, Palo Alto, USA, Junho 2000.

[LZ01] R.A. Lotufo e F.A. Zampirolli. Fast multidimensional parallel Euclidean distance transform based on mathematical morphology. No *Brazilian Symposium on Computer Graphics and Image Processing*, páginas 100–105, Florianopolis, Brasil, Outubro 2001.

[MR00] A. Meijster e J.B.T.M. Roerdink. A general algorithm for computing distance transforms in linear time. No *Mathematical Morphology and its Applications to Image and Signal Processing*, volume 18, páginas 331–340. Palo Alto, USA, Junho 2000.

[Rag92] I. Ragnemalm. Neighborhoods for distance transformations using ordered propagation. *Computer Vision, Graphics and Image Processing: Image Understanding*, 56(3), 1992.

[RP66] A. Rosenfeld e J.L. Pfalz. Sequential operations in digital picture processing. *Journal of the Association for Computing Machinery*, 13(4):471–494, Outubro 1966.

[RP68] A. Rosenfeld e J.L. Pfalz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.

[SC94] Y. Sharaiha e N. Christofides. Graph-theoretic approach to distance transformations. *Pattern Recognition Letters*, 15(10), 1994.

[Ser82] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[SM92] F.Y.-C. Shih e O.R. Mitchell. A mathematical morphology approach to Euclidean distance transformation. *IEEE Transactions on Image Processing*, 1:197–204, 1992.

[ST94] T. Saito e J. I. Toriwaki. New algorithms for Euclidean distance transformations of an n-dimensional digitized picture with applications. *Pattern Recognition*, 27:1551–1565, 1994.

[Vin92] L. Vincent. Morphological algorithms. No E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, capítulo 8, páginas 255–288. Marcel Dekker, New York, 1992.

[Yam84] H. Yamada. Complete euclidean distance transformation by parallel operation. páginas 69–71, 1984.

[Zam97] F.A. Zampirolli. Operadores morfológicos baseados em grafos de vizinhanças – uma extensão da MMach toolbox. Dissertação de Mestrado, Unversidade de São Paulo, São Paulo, Brasil, Abril 1997.

[ZL00] F.A. Zampirolli e R.A. Lotufo. Classification of the distance transformation algorithms under the mathematical morphology approach. No *Brazilian Symposium on Computer Graphics and Image Processing*, Gramado, RS, Brasil, Outubro 2000.