Comparação e desenvolvimento de algoritmos de transformada de distância euclidiana e aplicações

Ricardo Fabbri

Orientador: *Prof. Dr. Odemir Martinez Bruno*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC – USP, como parte dos requisitos para a obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP – São Carlos
Agosto/2004

A Ana Luiza Fabbri, que me trouxe alegria em momentos importantes.

Resumo

A Transformada de Distância (TD) é um operador geral que constitui a base de diversos algoritmos em visão computacional e geometria discreta, com grande poder de aplicação prática. No entanto, todos os diversos algoritmos ótimos para o cálculo da TD euclideana (TDE) exata surgiram apenas a partir da década de 1990. Não estava claro quais são os melhores algoritmos de de TDE exata, nem mesmo se realmente são exatos. Além disso, a implementação de tais métodos não é trivial e muitas vezes difícil de ser realizada eficientemente a partir da descrição nos artigos. Neste trabalho, são comparados experimentalmente e teoricamente os principais algoritmos de TDE, visando-se obter conclusões mais sólidas das diferenças de desempenho e exatidão de cada um. Os algoritmos também são descritos de maneira unificada e inédita nesta dissertação. Tais realizações são essenciais não só na teoria, mas também para viabilizar a aplicação prática dos algoritmos rápidos de TDE.

Abstract

The Distance Transform (DT) is a general operator forming the basis of many algorithms in computer vision and geometry, with great potential for practical applications. However, all the various optimal optimal algorithms for the computation of the exact Euclidean DT (EDT) were proposed only in the 1990 decade. Until now, it was not clear which are the best exact EDT algorithms, nor even if they are really exact. Moreover, their implementation is non-trivial and often difficult to perform efficiently using only the descriptions in the original papers. In this work, the main EDT algorithms are compared in theory and practice, in an effort to reach more solid conclusions of their differences in speed and their exactness. These realizations are essential not only in theory, but also to increase the applicability of bleeding-edge TDE algorithms.

Agradecimentos

Agradeço aos meus pais, em primeiro lugar, por serem meus grandes mentores e as melhores pessoas que conheço.

Obrigado à FAPESP, que custeou meu projeto de mestrado e de iniciação científica, bem como boa parte dos recursos que utilizei durante meus estudos na USP. Obrigado, também, ao CNPq por ter me incentivado com a minha primeira bolsa de iniciação científica.

Ao prof. Luciano da Fontoura Costa, que me guiou desde a iniciação científica por entre os problemas mais interessantes em processamento de imagens. Luciano é co-orientador deste trabalho. O agradeço imensamente, também, por ter-me concedido o uso dos ótimos recursos de seu grupo de pesquisa. Agradeço, de modo geral, o grande incentivo e apoio durante todos estes anos, que, em última instância, resultou na realização do meu grande sonho de fazer doutorado nos EUA.

Ao meu orientador prof. Odemir M. Bruno, por toda a sua energia, disposição, gentileza e competência. Obrigado por sempre ter acreditado no meu potencial de terminar este mestrado na metade do tempo. Sem a sorte de ter seu incentivo e garra, eu não teria chegado onde estou.

Ao prof. Alexandre X. Falcão, do IC–UNICAMP, por ter fornecido assistência valiosa na implementação da sua *Image Foresting Tranform*.

Ao meu grande amigo, Dr. Leandro F. Estrozi, pelas discussões, sugestões e críticas. Apesar de não ter se envolvido diretamente com o presente trabalho, trocamos idéias muito profícuas em assuntos correlatos, como diagramas de Voronoi pontuais ou eixos mediais. Obrigado também pela imagem de neurônio utilizada na Figura 1.2(a).

Aos meus irmãos do Grupo de Pesquisa em Visão Cibernética, sentirei muita falta de vocês.

Aos professores do ICMC–USP que sempre me apoiaram, tanto com incentivos sinceros como com cartas de recomendação. Muito obrigado por reconhecerem mais que justamente meu esforço e dedicação.

Sumário

Lista de Figuras

Lista de Algoritmos

Parte I

Revisão da Transformada de Distância Euclidiana

Capítulo 1

Introdução

1.1 Panorama do Trabalho

A transformada de distância (TD) mapeia cada ponto de uma imagem em sua menor distância a determinados subconjuntos dessa imagem [1]. Trata-se de um operador fundamental e com grande poder de aplicação, como evidenciado pela sua vasta literatura (ver bibliografia). Os métodos de TD são úteis principalmente por fornecerem uma representação e construção eficiente da propagação sucessiva de contornos, conhecida como evolução *eikonal* [2]. Esse tipo de propagação, por sua vez, está relacionada a diversas outras entidades em processamento de imagens, como eixos mediais, diagramas de Voronoi, determinação de caminhos mínimos e segmentação.

Várias métricas têm sido utilizadas para o cálculo da TD, como será descrito na Seção 1.4. A métrica euclidiana é necessária em várias aplicações, pois é o modelo adequado para inúmeros fatos geométricos, principalmente na escala da visão natural. Como ocorre na matemática, entretanto, algumas métricas não-euclidianas podem facilitar diversos cálculos e manipulações. De fato, já em 1966 foram relatados algoritmos eficientes para a TD utilizando métricas não-euclidianas [1]. Algoritmos eficientes de Transformada de Distância Euclidiana (TDE) exata, no entanto, surgiram apenas a partir da década de 1990, sendo que novos continuam sendo relatados, como será descrito no Capítulo 2.

Ainda não está claro qual é o melhor dos algoritmos de TDE exata, nem mesmo se estão corretos. A validação dos métodos realizada na literatura é escassa e incompleta. Na maioria dos casos, os testes de comparação e validação são realizados pelos próprios autores de algum método sendo julgado. Ademais, são escolhidos apenas alguns casos que podem esconder defeitos dos métodos e ressaltar apenas suas qualidades.

Essa falta de validação de algoritmos é típica na área de análise de imagens, principalmente pelo fato dos algoritmos serem bastante complexos e mesmo por uma relativa falta de valorização desse tipo de atividade [3]. Em particular, a validação da TDE é dificultada por vários fatores. Primeiro, os inúmeros algoritmos de TDE são relativamente elaborados tanto na teoria quanto na implementação. Segundo, o desempenho depende do conteúdo das imagens (e do tamanho,

claramente). Não é trivial prever o comportamento de um algoritmo de TDE para uma determinada imagem de entrada. Em terceiro lugar, existem vários fatores para comparar os algoritmos, como desempenho temporal, espacial, exatidão e facilidade de implementação. Por último, os testes realizados na literatura se mostraram insuficientes para uma comparação satisfatória dos algoritmos de TDE, atestando que a tarefa não é trivial.

1.2 Objetivos

O principal objetivo deste projeto é estudar, comparar e validar, experimentalmente e teoricamente, ao menos seis algoritmos de TDE, chegando-se a conclusões sólidas sobre as diferenças de desempenho e exatidão de cada um. Tais conclusões são essenciais não só na teoria, mas também para viabilizar a aplicação prática dos algoritmos rápidos de TDE. Esse estudo também tem potencial para tornar mais eficientes e confiáveis todas as diversas entidades relacionadas à TDE, como esqueletização, diagramas de Voronoi discretos, dimensão fractal, dentre outros.

A principal pergunta imediata a ser respondida pelos testes empíricos é:

- Quais algoritmos são comprovadamente exatos, quais os mais rápidos e mais fáceis de implementar?

Tal pergunta ainda não foi respondida. Existem ao menos seis algoritmos recentes para a TDE candidatos plausíveis a uma comparação [4, 5, 6, 7, 8, 9].

Também pretende-se extrair medidas das imagens de entrada para caracterizar a performance dos algoritmos de acordo com o a forma do objeto de entrada. Idealmente, deseja-se um método que aponte o melhor algoritmo para uma dada imagem. Se isto não for realizado automaticamente, tendo-se as medidas relevantes para a performance dos algoritmos o usuário poderá avaliar prever quais são os algoritmos satisfatórios para o seu problema. Obviamente, se um dos algoritmos se mostrar muito superior aos demais para todos os tipos de imagens, esta análise será desnecessária. Entretanto, resultados preliminares já mostram que os algoritmos de TDE avaliados possuem comportamento bastante variado.

Neste projeto também pretende-se incorporar aos principais algoritmos de TDE a propagação de rótulos e outras informações que auxiliem na solução de uma maior classe de problemas em análise de imagens, como será descrito na Seção 1.5.

1.3 Organização do Trabalho

Este trabalho está dividido da seguinte forma. Na próxima seção, são formalmente apresentadas as definições necessárias à compreensão do restante do texto. Entretanto, sugere-se que as definições da Seção 1.4.2 sejam lidas sob demanda, ou seja, apenas na medida em que forem utilizadas no texto.

A seguir, na Seção 1.5, realiza-se um levantamento de diversas aplicações de TD, de forma a ilustrar sua importância e sua relação com outras entidades. A Seção 1.6 ilustra a importância da

métrica euclidiana, motivando a busca por algoritmos eficientes de TDE. Os principais algoritmos recentes de TDE são organizados e explicados no Capítulo 2, já relatando o estudo teórico dos mesmos realizado pelo aluno. O Capítulo 3 apresenta a metodologia adotada neste trabalho para os testes empíricos de velocidade e exatidão. Os resultados dos testes são relatados no Capítulo 4. Porfim, o Capítulo 5 lista as principais contribuições do trabalho, bem como atividades a serem desenvolvidas futuramente.

1.4 Definições

1.4.1 Principais conceitos

O conceito de transformada de distância (TD) é simples, porém é necessário estabelecê-lo com rigor e clareza, pois na literatura se usam definições ligeiramente diferentes que podem causar confusão. Além disso, diversos conceitos e convenções necessitam estar precisamente definidos para a correta compreensão deste trabalho, principalmente para o Capítulo 2.

O problema central da TD é calcular a distância de cada ponto do plano a um determinado subconjunto deste. Em processamento de imagens, esse problema é refraseado da seguinte forma. Seja uma imagem binária $I : \Omega \subset \mathbb{Z}^2 \to \{0, 1\}$, onde, neste trabalho, o domínio Ω é convexo e, em particular, $\Omega = \{1, \ldots, n\} \times \{1, \ldots, n\}$ caso nada diferente tenha sido afirmado. Por convenção, 0 está associado a 'preto' e 1 a 'branco'. Tem-se, naturalmente, um objeto $\mathcal{O}$ representado por todos os pixels brancos:

$$\mathcal{O} = \{p \in \Omega \mid I(p) = 1\}$$

O conjunto $\mathcal{O}$ é denominado 'objeto' ou '*foreground*', podendo ser qualquer subconjunto do domínio da imagem, inclusive desconexo. Os elementos de seu complementar, $\mathcal{O}^c$, isto é, o conjunto de pixels pretos em Ω, são denominados 'fundo da imagem', '*background*', 'pontos de interesse', 'sementes', 'fontes', 'pontos de *feature*' (*feature points*), '*sites*', 'elementos de Voronoi' (EV), ou 'pixels pretos'. Apesar de contra-intuitivo à primeira vista, os 'pixels de interesse' realmente são os pixels de $\mathcal{O}^c$ para as definições utilizadas neste trabalho.

Definição 1 *A **transformada de distância (TD)** é a transformação que gera um mapa D cujo valor em cada pixel p é a menor distância desse pixel a $\mathcal{O}^c$:*

$$D(p) := min\{d(p, q) \mid q \in \mathcal{O}^c\} = min\{d(p, q) \mid I(q) = 0\} \tag{1.1}$$

*A imagem D é denominada **mapa de distâncias** de $\mathcal{I}$ (ou de $\mathcal{O}$, caso $\mathcal{I}$ esteja subentendida).*

D também pode ser chamada transformada de distância caso não haja ambiguidade entre a imagem D e a transformação que a gerou (TD). O termo TD também pode se referir a um algoritmo de TD, dependendo do contexto.

Assume-se que $\mathcal{O}^c$ contém ao menos um pixel [1]. Ademais, $d(p, q)$ é geralmente tomada

como a distância euclidiana, dada por:

$$d(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}. \tag{1.2}$$

A Figura 1.1 mostra um exemplo simples de cálculo da TDE. Para cada pixel em (a), o pixel

(a) **(b)**

Figura 1.1: Exemplo numérico de transformada da distância. Em (b), tem-se, para cada pixel, sua distância euclidiana ao pixel preto mais próximo. As distâncias são elevadas ao quadrado para que sejam valores inteiros.

correspondente na TD da figura (b) armazena a menor distância euclidiana entre esse pixel e todos os pixels pretos. Como as coordenadas dos pixels são números inteiros, geralmente trabalha-se com o quadrado da distância euclidiana, $d^2(p,q)$, que será também um número inteiro.

Pode-se visualizar a TD através de uma superfície com altura proporcional às distâncias, como na Figura 1.2(b), ou através de uma imagem com intensidade proporcional às distâncias (Figura 1.2(c)). Outra visualização interessante da TD pode ser obtida realizando-se a operação módulo n (resto da divisão por n) para cada valor de distância: $D_{mod\,n}(p) = D(p) \mod n$. Na medida em que a distância cresce, o valor de $D_{mod\,n}$ se repete, ficando sempre de 0 a $n-1$. Dessa forma, irão existir transições abruptas de $n-1$ a 0, o que facilita visualizar as curvas de isodistância.

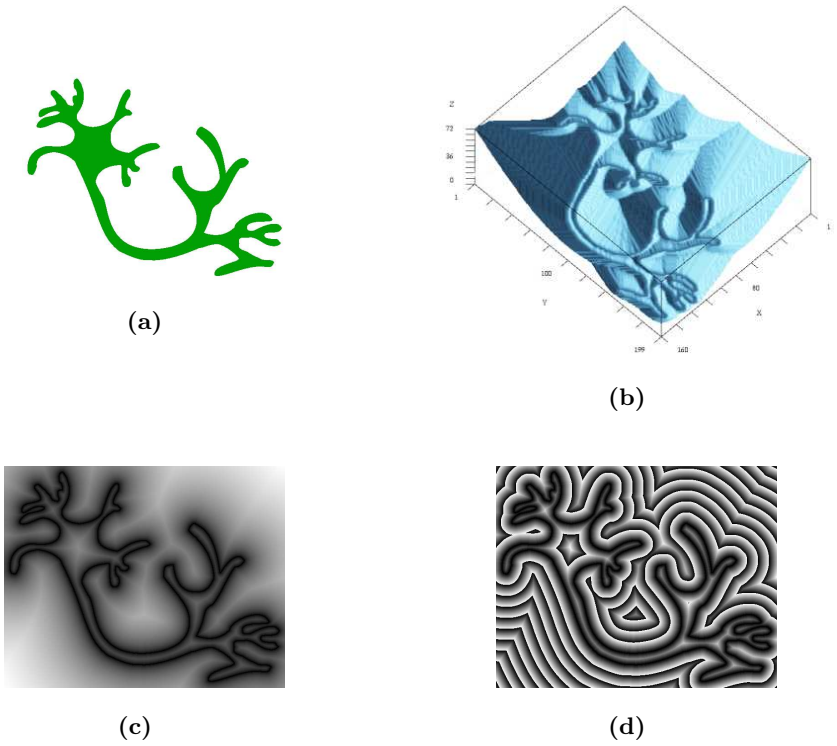
Várias métricas além da euclidiana podem ser utilizadas para calcular a distância na Equação 1.1. Como exemplos, têm-se as métricas *cityblock* (d_1) e *chessboard*(d_∞), definidas por:

$$d_1(\mathbf{x},\mathbf{y}) = |\mathbf{x}_1 - \mathbf{y}_1| + |\mathbf{x}_2 - \mathbf{y}_2|$$
$$d_\infty(\mathbf{x},\mathbf{y}) = max\{|\mathbf{x}_1 - \mathbf{y}_1|,\, |\mathbf{x}_2 - \mathbf{y}_2|\}$$

Tais métricas são menos custosas do que a métrica euclidiana.

1.4.2 Outras definições e convenções para referência

Neste texto, N denota o número total de pixels em uma imagem, e n denota o número de linhas ou colunas em uma imagem quadrada $n \times n$. Pode-se interpretar (e definir) n de forma mais geral, como sendo $\sqrt{N}$ para uma imagem $c \times r$ de $N = c \cdot r$ pixels, onde r e c são o



(a) **(b)** **(c)** **(d)**

Figura 1.2: A imagem de um neurônio (a) e representações da transformada de distância da sua borda, onde a altura da superfície (b) ou o brilho (c) são proporcionais à menor distância de cada ponto aos pixels de borda. Em (d) tem-se uma representação das curvas de isodistância.

número de linhas e colunas, respectivamente. O símbolo $\mathcal{N}$, diferentemente de n e N, denota uma vizinhança, definida como uma coleção de pares ordenados que representam vetores de deslocamento relativo.

Convenciona-se que $d(p, q)$ pode denotar a distância euclidiana ao quadrado ou a distância euclidiana com raiz, dependendo do contexto. A notação de norma também é muito utilizada: $\|p\| := d(p, \mathcal{O})$, onde $\mathcal{O}$ é a origem, e $d(p, q) = \|p - q\|$.

Uma TDE (i.e. algoritmo de TDE) é dita *exata* para um determinado conjunto de imagens se ela é exata para todas as imagens desse conjunto. Se for inexata para uma das imagens consideradas, a TDE é denominada *inexata*. Se a TDE for exata e o conjunto de interesse não for especificado, a TDE é considerada exata para toda imagem de entrada possível.

Durante este trabalho, será discutida a complexidade temporal dos algoritmos de TDE. O limite assintótico superior $O(f(n))$, limite assintótico inferior $\Omega(f(n))$ e limite assintótico equivalente $\Theta(f(n))$ serão expressos em função de n, e não de N.[1] Como qualquer algoritmo de TDE necessariamente visita uma vez cada pixel da imagem, o melhor algoritmo possível será $O(n^2)$, $\Omega(n^2)$ e, portanto, $\Theta(n^2)$. De fato, como será visto no Capítulo 2, diversos autores

[1] Para maiores informações sobre as notações $\Omega(f(n))$, $\Theta(f(n))$ e $O(f(n))$, recomenda-se [10]. A notação $\Theta(f(n))$ exprime um limite assintótico exato para $f(n)$, sendo descorrelacionada com a complexidade para o caso médio de um algoritmo.

propuseram algoritmos de TDE em tempo $\Theta(n^2)$. Como $N = n^2$, diz-se que tais métodos ótimos são lineares quanto ao número de pixels de entrada.

Outra convenção adotada é que pixel de fronteira (ou de contorno ou de borda) é um pixel branco que possui ao menos um vizinho preto. Fronteira (ou contorno ou borda) é o conjunto de todos os pixels de fronteira.

Outra entidade referenciada neste texto é o diagrama de Voronoi pontual (DV) e objetos relacionados. As definições adotadas neste texto são expostas brevemente a seguir. Para uma exposição mais didática e completa acerca de Diagramas de Voronoi, vide [11, 12, 13]. A região de Voronoi (RV) de um ponto de interesse[2] é o conjunto de pontos estritamente mais próximos deste do que qualquer outro ponto de interesse. Outros nomes para RV são zona de influência, polígono de Voronoi ou ladrilho (*tile*). Denomina-se $EV(p)$ o Elemento de Voronoi mais próximo a um determinado pixel p. Caso p for mais próximo a dois ou mais *sites*, escolhe-se um deles arbitrariamente como sendo $EV(p)$. Por definição, o diagrama de Voronoi pontual é o conjunto desses pontos mais próximos de dois ou mais *sites*, ou seja, fora de todas as regiões de Voronoi. *Partição de Voronoi* é a coleção das RVs de todos os *sites*; para montar a partição, cada ponto do DV é arbitrariamente atribuído à RV de um dos *sites* minimamente equidistantes a ele.

A partição de Voronoi pode ser representada pela transformada de rótulos (*label transform*), onde se imagina que cada EV possui um rótulo (número) associado que identifica unicamente este EV e sua RV correspondente. A transformada de rótulos pode ser definida formalmente por:

$$\begin{aligned} Rot : \Omega &\to \{1, \ldots, n_s\} \\ p &\mapsto Rot(p) = \{Rot(q) \mid q = EV(p)\} \end{aligned}$$

onde n_s é o número de *sites* (EVs).

Uma transformação bastante semelhante é a chamada transformada do *site* mais próximo, transformada de *feature*, que a cada pixel p associa $EV(p)$. Ainda outra entidade similar é a TD vetorial, que a cada pixel p associa um vetor apontando para $EV(p)$.

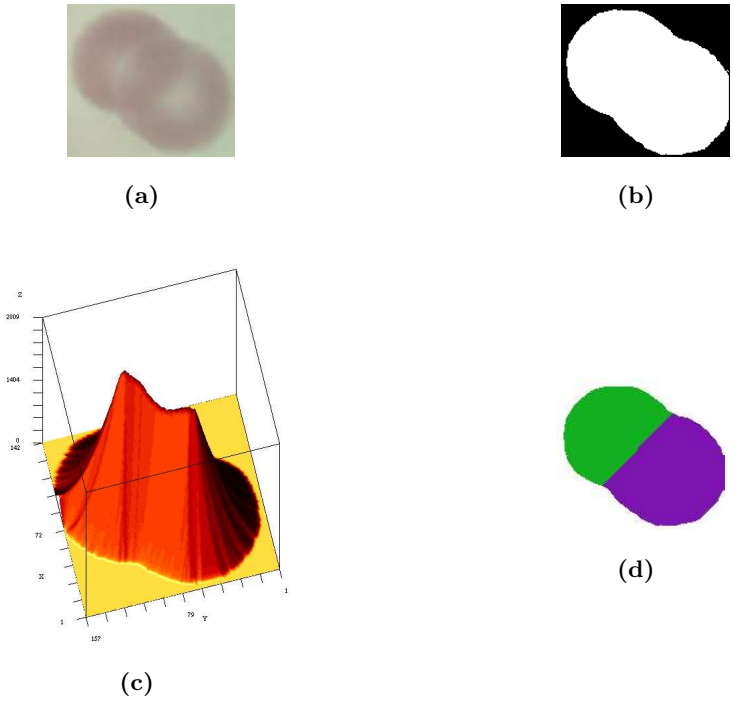
1.5 Aplicações

A TD é um operador fundamental em análise de formas, sendo utilizada em diversas aplicações, tanto práticas como teóricas. A seguir estão brevemente listadas algumas áreas de aplicação:

- **Separação de objetos sobrepostos**, através da segmentação por *watershed* [14, 6]. A Figura 1.3 ilustra brevemente esta aplicação. Suponha-se que se quer contar o número de hemácias em uma imagem microscópica. Um problema que pode ocorrer está ilustrado na Figura 1.3(a), onde se têm duas hemácias sobrepostas, que aparecem como um único componente conexo na imagem binarizada (Figura 1.3(b)). Para que a contagem de células seja correta, os casos de sobreposição devem ser separados. Uma maneira é utilizar a TD,

[2] lembre-se dos outros nomes para ponto de interesse são: *site*, Elemento de Voronoi (EV), semente e fonte

mostrada na Figura 1.3(c). Nota-se que os picos da TD estão nos centros dos dois glóbulos sobrepostos. Calcula-se em seguida a segmentação por *watersheds* do inverso da TD, resultando na separação dos glóbulos ilustrada na Figura 1.3(d).



(a) **(b)** **(c)** **(d)**

Figura 1.3: Separação de hemácias sobrepostas (a). Passos: (b) segmentação inicial; (c) transformada de distância euclidiana; e (d) imagem corretamente segmentada via *watersheds*.

- **Aplicação sucessiva de operadores morfológicos [15, 16]**. Se a TD for limiarizada em um nível r, obtém-se uma imagem cujos pixels de objeto são aqueles cuja distância ao plano de fundo é maior que r. Isso é o mesmo que erodir a imagem original por um disco de raio r. Portanto, uma vez calculada a TD, basta limiarizá-la para obter a erosão com o raio desejado. Similarmente, considerando-se a imagem invertida, obtém-se a TD do *background*. Limiarizando-a, obtém-se a dilatação da imagem original por um raio arbitrário. A TD é, por conseqüência, extremamente útil para implementar outras operações da morfologia matemática que são baseadas em erosões e dilatações sucessivas. Por exemplo, pode-se utilizá-la para representar em uma imagem operações sucessivas de abertura [16]. A partir dessa imagem pode-se gerar o chamado espectro de porosidade ou rugosidade (*roughness spectrum*) [16], aplicado estensivamente na classificação de formas de materiais porosos, em que ocorrem buracos (poros) de tamanhos variados. Um exemplo de tais imagens é proveniente de **reservas rochosas de óleo** [16]. A dilatação sucessiva também é útil para gerar um **espaço de escala morfológico [17]**, permitindo uma análise estrutural intrínseca e hierárquica da forma.

- A TD é utilizada para extrair muitas outras representações e medidas, como **esquele-**

tos [18, 19], diagramas de Voronoi [20], triangulação de Delaunay [20], dimensão fractal [21], grafos de Gabriel [20], dentre outros [22].

- **Navegação robótica**, para encontrar o caminho mínimo de um ponto a outro ou um caminho centralizado dentre obstáculos [23, 6].
- **Casamento de formas (*matching*) [24, 25, 26, 27]**. Basicamente, esta aplicação consiste na correlação de um *template* binário com a TD da imagem binária onde se está buscando objetos parecidos com o *template*. Uma vantagem de se utilizar a TD aqui é o fato da imagem de correlação resultante ser mais suave do que a correlação feita diretamente sobre a imagem original, sendo portanto adequada para a convergência rápida de algoritmos de otimização. Maiores detalhes podem ser encontrados no artigo de Paglieroni [25].
- **Medidas de forma relacionadas a distância [28, 1, 6]**. Por exemplo, o máximo da TD de um objeto é a sua largura; a distribuição das distâncias da TD também é um descritor bastante útil de uma forma.
- Outras áreas interessantes onde a TD também tem sido aplicada são: **ajuste ou registro de imagens (*image registration*) [29, 6] e visão estéreo [25], análise de imagens médicas [6, 30, 31, 32, 33, 7], análise de dados multi-dimensionais (classificação, *clustering*) [34], realce de imagens [35], otimização de *ray-tracing* [36], botânica [37]** e **geologia [36, 16]**.

Existem até mesmo evidências de que processos ligados à TDE estão relacionados a fenômenos de percepção e biologia [38, 39, 40, 41].

1.6 Importância da TD Euclidiana e sua Exatidão

Apesar de possuir uma definição simples e intuitiva, a TD é difícil de ser calculada com boa eficiência e precisão. O cerne da dificuldade consiste no fato da definição envolver uma minimização de distância pontual (ver Equação 1.1).

Uma maneira de aumentar a eficiência do cálculo da TD é aproveitar propriedades locais da métrica utilizada para evitar redundâncias no cômputo da minimização global para cada pixel, como explicado mais adiante no Capítulo 2. Esse tipo de otimização tem sido explorado como base de algoritmos eficientes para métricas não-euclidianas desde que a TD foi definida em 1966 [1, 28, 42, 24].

Entretanto, propriedades não-locais da métrica euclidiana em grades discretas [18, 6, 43], explicadas na Seção 2.3, levaram a dificuldades em se descobrir abordagens eficientes para TDE durante muito tempo. Algoritmos rápidos para o cálculo da TD euclidiana exata em computadores seqüenciais surgiram apenas a partir da década de 1990, sendo que novos continuam sendo publicados (cf. [4, 7, 8, 44, 5, 18, 45, 46, 43, 20, 47, 48, 15, 9]).

A métrica euclidiana possui diversas propriedades imprescindíveis em aplicações. Ela é radialmente simétrica, diferentemente das outras métricas. Isso permite gerar representações de

forma invariantes à rotação, o que é crucial para o reconhecimento. Esqueletos, ou eixos mediais, por exemplo, não são invariantes à rotação de um mesmo objeto quando métricas não-euclidianas são utilizadas. Conseqüentemente, nesse caso o reconhecimento de um objeto que sofreu uma rotação pode ficar comprometido. Ademais, com métricas não-euclidianas o caminho mínimo ou a largura máxima de um objeto podem não corresponder ao resultado efetivo verificado na prática. A métrica euclidiana também possui vantagens para o casamento de padrões, pois permite uma convergência mais rápida de métodos de otimização [6, 25, 24].

Métodos aproximados para a TDE têm sido propostos desde 1980 [19], fornecendo uma boa precisão e eficiência. No entanto, um mínimo erro na TDE pode acarretar conseqüências indesejáveis nas aplicações. Por exemplo, o esqueleto obtido a partir da TDE pode ficar desconectado [18] em vários casos comuns, violando uma propriedade crucial dessa representação. Ademais, o cálculo de caminhos mínimos fica comprometido, pois nesse caso o erro na solução é acumulado, podendo atingir níveis inaceitáveis. Outro problema significativo devido à inexatidão ocorre na implementação de operadores de morfologia [6].

Capítulo 2

Algoritmos de TDE

Nesta seção, os algoritmos de TD conceitualmente mais importantes e os mais eficientes serão descritos. Ênfase maior será dada aos diversos algoritmos euclidianos que formam o estado da arte na área [4, 5, 6, 7, 8, 9]. Um dos objetivos é apresentar um levantamento bibliográfico atualizado e abrangente, de maneira a reunir e organizar área de TDE, algo ainda pouco realizado, principalmente incluindo algoritmos recentes.

A descrição conceitual e uniformizada dos recentes algoritmos de TDE por si só constitui uma contribuição deste trabalho. Geralmente, os artigos originais descrevem os algoritmos de forma isolada, demasiadamente concisa e geral. Por exemplo, apresenta-se o novo método em n-dimensões, domínios não-convexos, métricas gerais ou grades não-ortogonais. Neste texto, diferentemente, o objetivo das descrições é apresentar a essência de cada método e superar as dificuldades encontradas durante a leitura e implementação dos originais. Um dos artifícios para se atingir esse objetivo é restringir os conceitos a imagens 2D e métrica Euclidiana, que são os casos mais comuns. Maiores detalhes e generalizações são encontrados nos artigos originais.

Os principais métodos aqui descritos estão implementados em C pelo autor desta monografia. Os códigos serão disponibilizados na forma de software livre, tanto na biblioteca C Animal (*An Imaging Library*)[49] como no *toolbox* para o Scilab SIP [50], ambos escritos pelo autor.

2.1 TDE por Força Bruta

A aplicação direta da Definição 1 conduz ao seguinte algoritmo de TDE: Para cada pixel p, calculam-se as suas distâncias a todos os pixels pretos; o mapa de distâncias em p é igual à menor dessas distâncias. Claramente, se p for preto, ele já possui seu valor final, que é a distância zero.

Mesmo para este método trivial, o número de operações depende não só do tamanho da imagem, mas também do conteúdo. Suponha-se que o número de pixels pretos seja k e, portanto, o número de pixels brancos seja n^2-k. O número de comparações é, por conseqüência, $k\cdot(n^2-k)$, onde $0 \leq k \leq n^2$. Essa função atinge o máximo em $k = n^2/2$ e, assim, o máximo número de operações é $n^2/2 \cdot (n^2 - n^2/2) = n^4/4 = O(n^4)$.

O número mínimo e não-nulo de operações ocorre para $k = 1$ ou $k = n^2$, isto é, quando há apenas um pixel branco ou preto. Logo, o método força-bruta é $\Omega(n^2)$. Com freqüência, entretanto, k é uma função linear de n. Isto ocorre para imagens obtidas amostrando-se um contorno contínuo, por exemplo. Neste caso, o número de operações é $\Theta(n^3)$. Em suma, o algoritmo força-bruta é $O(n^4)$ (pior caso), $\Omega(n^2)$ (melhor caso), e tipicamente em torno de $\Theta(n^3)$. Para o algoritmo força-bruta não ser $\Theta(n^4)$, é necessário utilizar duas listas de ponteiros, uma para os pixels pretos e outra para os brancos. Isso requer $n \cdot n$ ponteiros. Por outro lado, o algoritmo $\Theta(n^4)$ – simplesmente dois *loops* ambos percorrendo a imagem toda – pode ser realizado na própria imagem de entrada.

2.2 Algoritmos Eficientes para TDE

Para aumentar a eficiência de cálculo da TD, o princípio geral é explorar a 'redundância' ou 'localidade' de aspectos da métrica. A localidade do *site* mais próximo de cada ponto, por exemplo, é uma propriedade importante de métricas no plano contínuo:

Propriedade 1 *Para cada ponto p, existe outro ponto q, em uma vizinhança de p, com o mesmo* site *mais próximo. Em outras palavras, regiões de Voronoi contínuas são sempre conectadas no plano contínuo. Formalmente,*

$$\exists q \in \mathcal{N}(p) \mid EV(p) = EV(q), \quad \forall p \in \mathbb{R}^2, \forall \mathcal{N}(p) \tag{2.1}$$

onde $\mathcal{N}(p)$ denota uma vizinhança de p.

Se p e q são restritos a coordenadas inteiras (grade discreta), a Propriedade 1 ainda vale para diversas métricas. Tal fato permite o uso de operações baseadas em vizinhanças discretas para calcular a TD para tais métricas.

Uma propriedade relacionada que permite deduzir a distância de um pixel a partir da distância de seus vizinhos é chamada *regularidade*:

Propriedade 2 *[51, 43] Uma métrica d é* regular *se, para todo p e q tal que $d(p,q) \leq 2$, existe um r, diferente de p e q, tal que $d(p,q) = d(p,r) + d(r,q)$*

Rosenfeld et. al. [1] foram os primeiros a propor algoritmos eficientes de TD, baseados em operações seqüenciais locais. Outros métodos deste tipo também foram propostos, como será revisado a partir da Seção 2.4.

2.3 Erros no Cálculo da TDE

A maioria dos algoritmos locais não calcula a TDE exata, pois as Propriedades 1 e 2 não valem para a métrica euclidiana em grades discretas. O *site* mais próximo de um pixel p pode ser diferente do *site* mais próximo de todos os seus vizinhos discretos para esta métrica. Tal fato é expresso em outras palavras na Propriedade 3, enunciada a seguir.

Propriedade 3 *Regiões de Voronoi euclidianas discretas podem ser desconectadas.*

A Propriedade 3 é a principal razão pela qual os primeiros algoritmos de TDE exata apareceram apenas nos anos 1990, enquanto algoritmos não-euclidianos eficientes apareceram desde 1966.

Exemplos de regiões de Voronoi com mais de um componente conectado-de-4 e conectado-de-8 são mostrados na Figura 2.1.

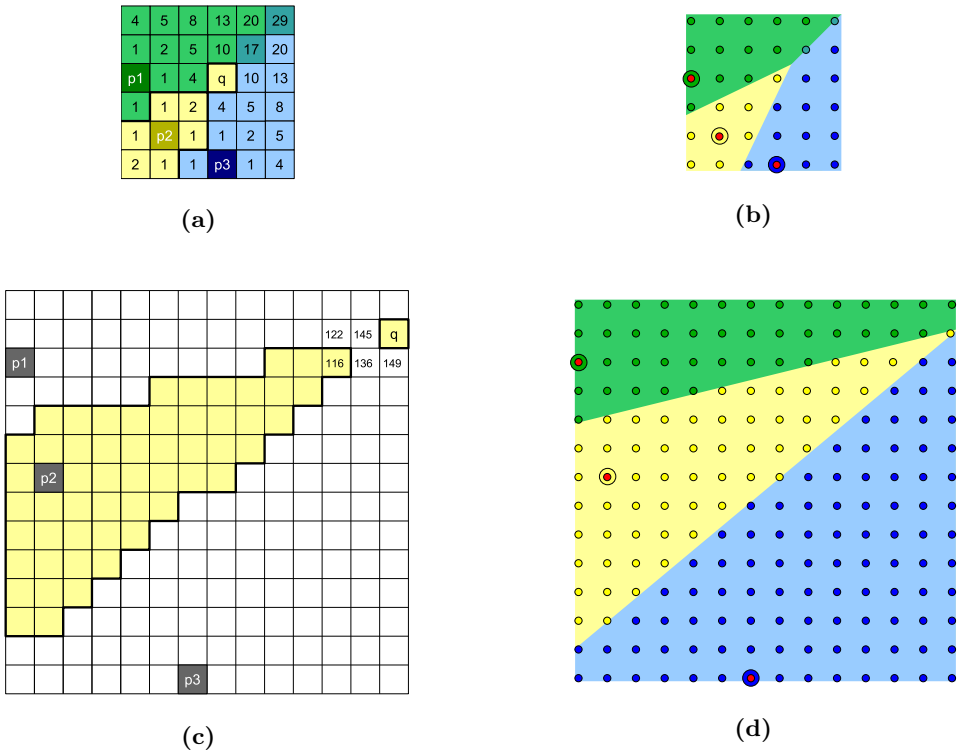


Figura 2.1: Regiões de Voronoi euclidianas desconexas para vizinhança de 4, (a) e (b), e vizinhança de 8, (c) e (d). O pixel q recebe um valor errado de distância se os algoritmos assumirem conectividade de 4 (a) ou de 8 (c) para as regiões discretas. Em (b) e (d), têm-se em cores mais claras as regiões contínuas e conexas, com os pontos de amostragem sobrepostos. Pixels equidistantes a p_1 e p_2 ou p_1 e p_3 estão em azul turquesa.

Na Figura 2.1(a), a RV do pixel p_2 é desconectada-de-4. O pixel q é mais próximo de p_2 do que de p_1 e p_3, porém nenhum de seus 4-vizinhos é mais próximo de p_2. Para ilustrar o motivo da RV desconexa, a Figura 2.1(b) mostra as regiões de Voronoi contínuas, em cores mais claras, juntamente com os pontos de amostragem, em cores mais fortes. Cada cor corresponde ao pixel mais próximo de cada ponto no plano contínuo. Os pixels equidistantes a p_1 e p_2 estão mostrados em azul turquesa. As Figuras 2.1(c) e 2.1(d) mostram que um problema similar ocorre para uma vizinhança maior 3×3. O problema de desconexão geralmente ocorre devido à amostragem de RVs finas e inclinadas que não passam por nenhum vizinho discreto de um determinado pixel.

A Propriedade 3 é causa de erro em mapas de distância gerados por uma classe de TDs definida a seguir.

Definição 2 *Denomina-se $\mathcal{N}$-TDE qualquer algoritmo de TDE cuja distância final de cada pixel é calculada em relação à mesma fonte de algum pixel em uma vizinhança $\mathcal{N}$ fixa. O mapa gerado por uma $\mathcal{N}$-TDE é denotado por $D_{\mathcal{N}}$, e o mapa gerado por uma TDE exata é denotado simplesmente por D.*

Exemplos de $\mathcal{N}$-TDE são os algoritmos de Danielsson [19] e o PSN [6, 43, 52]. Não é surpreendente o fato de que tais algoritmos apresentem erro devido à Propriedade 3. Por exemplo, na Figura 2.1, se o pixel mais próximo a q for inferido a partir de seus vizinhos, ocorrerá uma distância errada em q, pois nenhum deles possui o mesmo pixel mais próximo a q. De fato, a distância em q será calculada em relação a um pixel de interesse diferente daquele que é o mais próximo de q. Portanto, fica claro que $D_{\mathcal{N}}(q) \geq D(q)$, para todo pixel q.

Mostra-se que toda $\mathcal{N}$-TDE é inexata para todos os pixels com distância exata maior que um determinado valor $d_c(\mathcal{N})$. Esse valor depende apenas da vizinhança $\mathcal{N}$ utilizada. Formalmente:

Propriedade 4 *(baseada em [6, 43]) Dada uma vizinhança $\mathcal{N}$, existe um valor de distância $d_c(\mathcal{N})$ tal que:*

$$D_{\mathcal{N}}(p) \geq d_c(\mathcal{N}) \Rightarrow D_{\mathcal{N}}(p) > D(p),$$

para algum pixel p de alguma imagem.

Assumindo-se que o erro de desconexão seja a única fonte de erro das $\mathcal{N}$-TDEs, pode-se estender a Propriedade 4, gerando a Propriedade 5.

Propriedade 5 *(baseada em [6, 43]) Dada uma vizinhança $\mathcal{N}$, existe um valor $d(\mathcal{N})$ tal que, para todo pixel p:*

$$D_{\mathcal{N}}(p) < d(\mathcal{N}) \Rightarrow D_{\mathcal{N}}(p) = D(p)$$

Existe um valor maximal $d_c(\mathcal{N})$ para $d(\mathcal{N})$ a partir do qual $D_{\mathcal{N}}(p)$ é inexata para algum p e alguma imagem satisfazendo $D(p) \geq d_c(\mathcal{N})$. Além disso, quanto maior $\mathcal{N}$, maior $d_c(\mathcal{N})$.

Um majorante para $d_c(\mathcal{N})$ pode calculado através de um algoritmo de busca exaustiva descrito na página 60 da tese de Cuisenaire [43]. A única implementação aberta desse cálculo, que se tem notícia, é aquela realizada pelo o autor desta monografia. O código estará disponível na biblioteca Animal [49] assim que um artigo a respeito seja publicado.

A Propriedade 5 indica que, dada uma imagem, sua TDE exata pode ser calculada por uma $\mathcal{N}$-TDE com $\mathcal{N}$ suficientemente grande. Entretanto, a aplicação direta desta idéia é ineficiente para imagens grandes, pois o tamanho da vizinhança necessária pode ser muito grande. Ao considerar o número de elementos m da vizinhança empregada para cada tamanho de imagem, tem-se que a complexidade de uma $\mathcal{N}$-TDE ótima passa a ser $O(N \cdot m)$, onde N é o número de pixels da imagem.

2.4 Tipos de Algoritmos de TDE

Os Algoritmos eficientes e seqüenciais para TD podem ser classificados de acordo com a ordem em que os pixels são processados. Nos algoritmos de *propagação*, a informação de menor distância é calculada a partir dos pixels de interesse e transmitida para o restante da imagem em ordem crescente de distâncias. Já os algoritmos de *varredura raster* (*raster scanning*) utilizam máscaras 2D para processar a imagem linha por linha, de cima para baixo, esquerda para direita, e em seguida o inverso. Algumas variações deste processo existem, mas todos algoritmos nesta classe utilizam máscaras locais, discretas e 2D em cada passada na imagem. Os métodos de *varredura independente* (*independent scanning*) processam cada linha da imagem, independentemente uma da outra, e em seguida processam cada coluna do resultado. Este processo é similar à convolução gaussiana e transformada de Fourier separáveis [53]. Essa classificação bastante conhecida de algoritmos de TD é similar àquela usada para algoritmos de morfologia matemática [54].

As classes de TDs não são exclusivas, especialmente as de varreduras raster e independente. Vários algoritmos de varredura independente foram originados de métodos de *raster-scanning* relacionados, freqüentemente através da decomposição de uma operação 2D em transformações unidimensionais separáveis. Ademais, todos os algoritmos de TDE, em última instância, realizam propagação, porém apenas a classe de propagação ordenada simula uma propagação simultânea em todas as direções.

A seguir, os conceitos e algoritmos de cada classe são revisados, com ênfase nas TDs euclidianas.

2.4.1 Algoritmos de propagação: conceitos fundamentais

Uma maneira de calcular a TD é ilustrada pela analogia do incêndio de gramado ou *grassfire*. Imaginam-se campos gramados representados por uma imagem binária, da seguinte forma: 1 representa grama, 0 representa uma região sem grama. Suponha-se que um incêndio é iniciado na fronteira dos gramados. Na medida em que a grama é queimada, o fogo caminha progressivamente mais distante da sua posição inicial até que se extingue. Em outras palavras, em um tempo t, o fogo estará à distância d das regiões inicialmente sem grama. Marcando-se a distância da frente de fogo em cada pixel por onde ela passa, produz-se a TD da imagem binária que representa os campos gramados.

A analogia *grassfire* constitui a idéia básica por trás dos algoritmos de propagação ordenada. Iniciando-se dos pontos de fronteira, eles calculam as distância de pixels cada vez mais distantes. O processamento é mantido na faixa estreita de pixels (frente de fogo) onde alguma mudança pode ocorrer. No entanto, o desafio consiste em utilizar essas idéias para construir um algoritmo exato e eficiente para a métrica euclidiana.

O processo fundamental que realiza a propagação ordenada comum aos métodos desta classe é descrito no Algoritmo 1, listado a seguir.

O emprego de um conjunto de contorno restringe o processamento apenas à faixa estreita de pixels onde as distâncias têm potencial para mudar. Essa é a principal razão pela qual este

Algoritmo 1 Procedimento fundamental de TDs por propagação ordenada

1. Inicialize a distância de todo pixel branco para um valor suficientemente alto.
2. Inicialize um conjunto auxiliar de pixels, denominado *Conjunto de Contorno*, para armazenar os pixels de fronteira.
3. Enquanto o Conjunto de Contorno não está vazio, faça:
   (a) Remova um pixel do Conjunto de Contorno, denominado *pixel central*.
   (b) Para cada vizinho branco do pixel central, faça:
      i. Calcule uma nova distância para o vizinho, baseando-se na distância do pixel central.
      ii. Se esta distância nova for menor que a distância corrente do vizinho:
         - atualize sua distância corrente como sendo a menor.
         - coloque esse vizinho no Conjunto de Contorno.

procedimento pode ser eficiente. Note-se que nada foi dito sobre *qual* pixel deve ser removido pelo Passo 3a, ou *qual* a vizinhança a ser utilizada no Passo 3b. Basicamente, os detalhes destas etapas controlam a eficiência e a corretude do método. Vários pixels podem ser atualizados desnecessariamente se tais passos não forem projetados adequadamente.

O Algoritmo 1 é similar ao algoritmo de Dijkstra [55, 56, 57]. Em Dijkstra, o Passo 3a remove, do Conjunto de Contorno, o pixel com a menor distância atual. Nesse caso, o Conjunto de Contorno é uma fila de prioridades – uma estrutura em que o pixel com menor distância pode ser determinado rapidamente. A fila de prioridades é o gargalo mais importante do algoritmo de Dijkstra. Sua eficiência pode ser melhorada para TDs pois, em imagens digitais, os custos (distâncias) são inteiros limitados (e.g. métrica Euclidiana ao quadrado). Neste caso, é possível aplicar as otimizações de Dial [58] e Ahuja et. al. [59] para o algoritmo de Dijkstra, utilizando *buckets* para a fila de prioridades. Esta é a base para diversos algoritmos rápidos a serem explicados mais adiante. Trata-se também da base de métodos numéricos para resolver a equação diferencial parcial *Eikonal* – os métodos *fast marching* – os quais não serão abordados em profundidade neste trabalho.

A fila de *buckets* é um vetor indexado por um valor inteiro de distância. Cada *bucket* i contém uma lista que armazena os pixels com distância atual $d = i$. A vantagem dessa estrutura é que os pixels na fila estão naturalmente ordenados, da mesma forma como ocorre no algoritmo de *bucketsort*[60]. Para determinar um pixel com menor custo no Passo 3a, simplesmente incrementa-se a distância atual (índice do *bucket*) até se atingir um *bucket* não-vazio.

O Algoritmo 1 para a métrica euclidiana, cuja fila de prioridades é implementada com *buckets*, e cuja vizinhança utilizada é $n \times n$, é denominado **PSN** (*Propagation by a Single Neighborhood*). O tamanho da vizinhança afeta fortemente a exatidão dos resultados.

Como visto na Seção 2.3, para cada tamanho de vizinhança, existe uma distância até a qual a TDE gerada pelo PSN será exata. Para além desta distância, podem haver erros similares ao

da Figura 2.1. Cuisenaire propôs um método que corrige o PSN utilizando propagações com vizinhanças crescentes apenas em determinados locais, como explicado na Seção 2.9.

Existem outras otimizações aplicáveis aos algoritmos de propagação ordenada. Primeiro, a orientação da vizinhança pode ser restrita para reduzir atualizações desnecessárias. Isto é feito em [61, 62, 15, 5, 6] mas não em [52]. Segundo, a fila nunca armazena mais do que uma certa diferença máxima de custo a cada momento. Portanto, o *bucket* pode ser circular [52, 58], sendo o índice igual ao valor da distância *modulo* o máximo incremento distância possível.

Os algoritmos de propagação ordenada pela distância euclidiana possuem a propriedade importante de que a TDE pode ser calculada apenas até uma certa distância. Isto é particularmente adequado para implementar erosão e dilatação por um disco de raio r. Ademais, os esquemas de propagação são mais eficientes para domínios não-convexos [43].

2.4.2 Algoritmos de propagação: um breve levantamento histórico

Montanari [61] foi o primeiro trabalho a aplicar uma idéia similar ao Algoritmo 1 com *bucketsort*, para computar a TD e o eixo medial com uma métrica que aproxima a euclidiana. Ele também mostrou um resultado relevante sobre a direção em que a propagação deve ocorrer para minimizar o desvio da métrica euclidiana.

Em 1987, Piper e Granum [63] propuseram uma estratégia FIFO (*first-in-first-out*) para o Conjunto de Contorno: o Passo 3a simplesmente remove o pixel que está há mais tempo na fila. Trata-se de uma simples busca em largura. Este esquema pode fazer com que vários pixels sejam atualizados mais que uma vez, alguns sendo atualizados até uma vez por pixel de interesse [15]. Os autores também consideraram domínios não-convexos.

Verwer [62], em 1989, propôs um método de propagação para algoritmos não-euclidianos que também utiliza o Conjunto de Contorno com *buckets*. Ragnemalm [15] estendeu o trabalho para a métrica euclidiana. Verwer argumentou que esse tipo de propagação é mais apropriado do que varredura raster, para TDs restritas a domínios não-convexos.

Em seu trabalho de 1992, Ragnemalm [15] utiliza uma propagação circular por limiarização, mais uma estratégia para o Passo 3a do Algoritmo 1. Ele introduz uma variável limiarizadora que armazena uma cota superior para o valor de distância de todos os pixels a serem processados na vizinhança do Conjunto de Contorno atual. Os pixels com distâncias maiores que esse valor continuam no Conjunto de Contorno para a próxima iteração. Após cada iteração, essa cota superior é aumentada pelo máximo incremento implicado pela vizinhança utilizada. Ragnemalm afirma que este esquema remove quase todas as atualizações múltiplas, como mostrado por alguns experimentos [15]. No entanto Cuisenaire [43] afirma que alguns pixels podem ser atualizados várias vezes, atingindo uma vez por pixel de interesse em alguns casos, o que implica uma complexidade de $O(n^3)$. A razão para essa atualização múltipla, como acontece com Eggers [5], é que a propagação de Ragnemalm é ordenada pela métrica *chessboard* em vez da euclidiana [43].

Sharaiha e Christofides, em 1994 [64], propuseram uma TD por propagação, explicitamente formulada com teoria dos grafos e resolvida por uma variação do algoritmo de Dijkstra-Moore-Dial. Essa idéia foi recentemente estendida para a métrica Euclidiana e generalizada para

outros problemas de análise de imagens, formando uma abordagem unificada denominada *Image Foresting Transform* (IFT) [52]. A IFT formula diversos conceitos de análise de imagens como problemas de grafos e, assim, fornece a eles uma solução eficiente única – o algoritmo de Dial – e uma teoria unificada. A Seção 2.10 explica os principais conceitos da IFT e o cálculo da TDE por essa abordagem. A TDE por IFT publicada até a presente data é equivalente ao algoritmo PSN e, portanto, não é exata. Porém seu algoritmo é prontamente aplicável ao cálculo de diversas entidades relacionadas à TDE.

Neste trabalho, serão avaliados duas TDEs recentes por propagação ordenada: o método de Eggers [5] e Cuisenaire [6]. O método de Cuisenaire está descrito na Seção 2.9.

2.4.3 Algoritmos de varredura raster

Rosenfeld e Pfaltz propuseram os primeiros algoritmos seqüenciais de TD por varredura raster e métricas não-euclidianas [1, 28]. Em [28], eles propuseram métricas *cityblock*, *chessboard*, hexagonais e octogonais para aproximar a TD euclidiana.

Muitos autores melhoraram a idéia de varredura raster para melhor aproximar a métrica euclidiana com pouco *overhead* [61, 42, 24, 65]. O trabalho de Borgefors [42] revisa esses algoritmos e propõe as TDs *chamfer* (ou 'chanfradas'), amplamente utilizadas na prática. As métricas *chamfer* são regulares e definidas por máscaras locais. Os pesos das máscaras são escolhidos de forma a minmizar o desvio da TDs euclidiana. As TDs *chamfer* necessitam de uma varredura raster de duas passadas, da mesma forma que o algoritmo original de Rosenfeld e Pfaltz. Maragos e outros propuseram melhorias às TDs *chamfer* de Borgefors [65].

Em 1980, Danielsson [19] propôs um algoritmo para gerar a TD euclidiana baseado em uma idéia similar à varredura raster das TDs *chamfer*. No entanto, as informações propagadas são os valores absolutos das coordenadas relativas do pixel de interesse mais próximo, em vez de apenas distâncias relativas. Para tanto, o método utiliza dois valores nas máscaras, em vez de um, o que é chamado de propagação vetorial. Dadas essas coordenadas, a distância euclidiana é facilmente calculada.

Danielsson afirmou que, apesar da sua nova TDE ser correta para a maioria dos pixels, alguns erros são produzidos. Para melhorar a precisão, ele propôs o uso de máscaras maiores. Seu algoritmo com máscaras de vizinhança-de-4 é chamado 4SED (*Sequential Euclidean Distance map*) e o método mais preciso que utiliza máscaras de vizinhança-de-8 é chamado 8SED. Na Seção 2.5 o algoritmo de Danielsson será explicado em maiores detalhes, dada sua importância.

Existem diversos aprimoramentos sobre o algoritmo de Danielsson. Ye [18] propôs o *mapa de distâncias com sinal*, significando que as informações propagadas são as coordenadas relativas com sinal, e não seus valores absolutos como no SED original. Estes algoritmos são denominados 4SSED(*Signed* SED) e 8SSED.

Leymarie [66] propôs uma implementação eficiente do algoritmo de Danielsson, comparável às TDs *chamfer* mais rápidas. Outro aprimoramento importante foi publicado por Ragnemalm [67] – uma implementação do 8SSED com apenas 3 varreduras separáveis (i.e. sem varrer para frente e para trás em cada linha). Ele também mostrou que este é o menor número possível de varreduras.

A maioria das TDE exatas por varredura raster são baseadas em correções sobre alguma variação do SED. A correção exata de Mullikin [48] sobre o 4SED é considerada por Cuisenaire [43] como $O(n^3)$ no pior caso. Outro método, proposto por Shih e Liu [26], realiza o pós-processamento do 8SED para corrigir erros. No entanto, estes erros foram sub-estimados, como apontado por Cuisenaire [43].

Dois métodos recentes parecem ser os algoritmos de TDE exata mais rápidos de varredura raster [68, 9]. O primeiro, proposto por Cuisenaire, consiste em realizar um pós-processamento do 4SED. A idéia principal é aplicar certas operações em pixels da borda das regiões de Voronoi geradas pelo 4SED. Nesses pixels, um método simples é proposto para verificar se a RV contínua gerou pixels desconectados na grade discreta. Isto é realizado definindo-se fronteiras que incluem a RV discreta e limita o número de pixels a serem verificados.

O segundo método recente de TDE que afirma ser exato consiste em duas varreduras utilizando uma vizinhança 3×3, proposto por Shih e Wu [9]. Os autores supostamente provam a corretude do algoritmo e mostram que a complexidade é independente do conteúdo da imagem. Entretanto, testes informais realizados durante este mestrado indicam que esse método pode estar errado. A comprovação formal deste fato foi deixada para trabalho futuro, como indicado na Seção 5.2

Apesar das TDEs por varredura raster parecerem ser as mais rápidas, existem algumas desvantagens. Primeiramente, muitos pixels podem ser processados várias vezes, especialmente pelo SED, como apontado por Ragnemalm [15]. Os algoritmos eficientes de propagação, por outro lado, devem evitar essas atualizações e processar apenas os pixels necessários – o que pode resultar em menos de duas passadas na imagem. Em segundo lugar, esses algoritmos são mais difíceis de estender para domínios não-convexos.

2.4.4 Algoritmos de varredura independente

Rosenfeld e Pfaltz inventaram outra abordagem para calcular TDs com métrica *cityblock*, denominada varredura independente ou redução de dimensionalidade. A TD 1D é construída para cada linha (ou coluna) independentemente, e em seguida esse resultado é utilizado em uma segunda fase para construir a TD 2D completa.

O primeiro passo é similar a todas as TDEs baseadas em varredura independente:

Transformação 1 *Dada uma imagem de entrada F, a 1^a transformação dos algoritmos de varredura independente gera uma imagem G definida por:*

$$G(i,j) = \min_y \left\{ (j-y)^2 \mid F(i,y) = 0 \right\} \tag{2.2}$$

Isto corresponde a calcular, para cada pixel (i,j), sua distância ao pixel preto mais próximo na sua linha (ao quadrado). Esta transformação é implementada eficientemente fazendo-se uma varredura para frente (i.e. da esquerda para a direita) seguida de uma varredura para trás (i.e. da direita para a esquerda) em cada linha da imagem de entrada.

(a) **(b)**

Figura 2.2: Exemplo da transformação 1D comum às TDEs de varredura independente. Tem-se a imagem de entrada F em (a) e sua TDE 1D G ao longo das linhas em (b).

O resultado da *transformação 1* para uma imagem hipotética está ilustrada na Figura 2.2.

O processamento não-trivial consiste na segunda etapa; é nela que cada algoritmo aplica sua estratégia para gerar a TDE 2D a partir da transformada 1D de cada linha da imagem.

A abordagem de varredura independente foi generalizada por Paglieroni para a métrica euclidiana [25, 69], além de uma classe bastante ampla de métricas que satisfazem certas propriedades. Basicamente, essas propriedades importantes asseguram ser possível compor a TD 2D utilizando varreduras 1D independentes em cada direção. O segundo passo do método de Paglieroni consiste em varreduras em cada coluna, de baixo para cima e de cima para baixo, juntamente com testes para restringir o número de pixels de interesse a considerar para cada pixel. No entanto, o algoritmo é $O(n^3)$.

Propriedades particulares da métrica euclidiana foram exploradas por métodos subseqüentes para restringir o número de operações para cada pixel durante a segunda fase da varredura. Baseando-se nessas três propriedades, existem três sub-variantes de algoritmos de varredura independente. Uma variante utiliza propriedades baseadas em interseção de parábolas; outra utiliza uma abordagem de morfologia matemática; e a última classe se baseia no cálculo eficiente de interseções do diagrama de Voronoi com as linhas da imagem.

A idéia básica da eficiência das TDEs por varredura independente é que apenas $O(n)$ pixels-semente influenciam o mapa de distância ao longo de cada linha da imagem. Portanto, sabendo-se quais são esses pixels relevantes para cada uma das n linhas, a TDE completa é calculável em $n \cdot O(n) = O(n^2)$ operações. A determinação dos pixels relevantes para cada linha pode ser realizada em tempo $O(n)$, como mostrado por alguns artigos revisados adiante. Portanto, em princípio, dois passos $O(n)$ são efetuados para cada linha – a determinação dos sites relevantes e o cálculo da distância Euclidiana a partir dessa informação. Ademais, diversos métodos entrelaçam as duas etapas $O(n)$ para melhorar a performance não-assintótica.

Métodos baseados em interseções de parábolas

O algoritmo mais antigo desta categoria [70] é $O(n^2 log(n))$. Chen e Chuan [71] melhoraram o método, tornando-o linear no pior caso [72]. As propriedades utilizadas pelos autores para

melhorar a eficiência da segunda etapa são basesdos em interseção de parábolas.

Saito e Toriwaki [7] também propuseram um método que utiliza idéias similares, como será explicado na Seção 2.6. Apesar de ser rápido para diversas imagens, a complexidade desse método ainda não foi avaliada. Alguns autores afirmam que é $O(n^3)$ [8, 6, 43], baseados em observações experimentais não relatadas.

Algoritmos similares de TDE foram propostos mais recentemente por Meijster e Hirata [72, 73]. Segundo a opinião informal de certos autores [74], o método de Meijster é bem mais rápido que o de Saito.

Abordagem de Morfologia Matemática

Basicamente, Shih e Mitchell [75] mostraram que a TDE pode ser calculada por uma única erosão morfológica em níveis de cinza da imagem de entrada. Mais tarde, Huang e Mitchell [76] decompuseram a função estruturante em uma seqüência de funções estruturantes 3×3. Lotufo e Zampirolli [4] melhoraram este método decompondo ainda mais o elemento estruturante em elementos 1D. Como resultado, o algoritmo deles realiza varreduras independentes, sendo a primeira etapa similar às dos demais algoritmos deste tipo. A segunda etapa é realizada utilizando-se FIFOs para a propagação 1D. Mais detalhes são dados na Seção 2.8 e nos trabalhos [4, 74, 77].

Métodos baseados em interseções do diagrama de Voronoi

A TDE pode ser calculada utilizando os algoritmos tradicionais de diagrama de Voronoi [13, 78], que executam em tempo $O(n^2 logn)$. Entretanto, essa eficiência pode ser reduzida a $O(n^2)$, já que os sites da imagem têm coordenadas inteiras [47, 79].

Um fato importante é que a interseção do DV dos pixels-semente com uma linha ou coluna da imagem pode ser eficientemente calculada. Breu et. al.[47] propuseram a primeira TDE utilizando a construção dessas interseções, e provaram que o método é $O(n^2)$ e exato. Guan e Ma [80] melhoraram este método utilizando propriedades da métrica euclidiana e mantendo uma representação ligeiramente diferente para a interseção do DV com cada linha da imagem.

O recente método de Maurer et. al. é uma melhoria dos dois métodos anteriores, como será explicado na Seção 2.7. Sua corretude e complexidade temporal são provadas formalmente.

2.5 O Algoritmo de Danielsson

O algoritmo de TDE de Danielsson [19] é um dos mais utilizados, por ser eficiente e simples. Entretanto, não é exato. Dada uma imagem binária, o algoritmo gera sua TDE vetorial (ver Seção 1.4.2) em 4 varreduras raster, utilizando as máscaras com elementos multivalorados mostradas à esquerda na Figura 2.3.

O mapa de distância deve ser inicializado da seguinte forma: o pixel $I(i, j) = (0, 0)$ se for preto, e $I(i, j) = (\infty, \infty)$ caso contrário. O valor ∞ é um número maior que a máxima distância possível na imagem, geralmente o tamanho da diagonal mais um. Iniciando do topo, o mapa

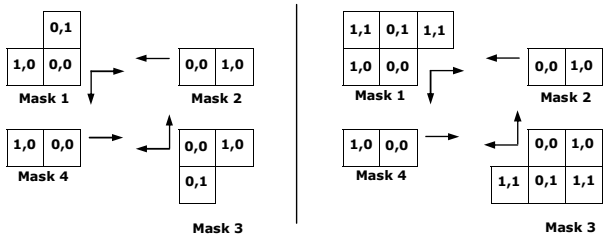


Figura 2.3: Máscaras dos métodos 4SED (à esquerda) e 8SED (à direita)

é varrido linha por linha movendo-se a máscara 1 da esquerda para a direita na linha atual e a máscara 2 da direita para a esquerda na mesma linha. Em cada pixel branco, os vetores da máscara são adicionados aos vetores correspondentes no mapa de distância e o novo valor é definido como o mínimo dessas somas. Em seguida, as máscaras 3 e 4 são movidas linha por linha da direita para a esquerda e da esquerda para a direita, respectivamente.

Este algoritimo é denominado 4SED (*Four-point Sequential Euclidean Distance transform*). O nome se refere ao fato de que os vizinhos 4-conectados são utilizados pelas máscaras. Apesar deste algoritmo gerar a TDE sem grandes erros, ele não é exato. Isto se deve à Propriedade 3, vista na Seção 2.3. Para reduzir os erros produzidos pelo 4SED, pode-se utilizar máscaras maiores, como mostradas à direita, na Figura 2.3. Usando essas máscaras, o algoritmo é chamado 8SED. Entretanto, ele também não é exato para distâncias suficientemente grandes. De fato, como vimos na Seção 2.3, para qualquer tamanho de vizinhança é possível encontrar um valor de distância para o qual um erro pode ocorrer na TDE.

2.6 O Algoritmo de Saito

Saito e Toriwaki [7] desenvolveram um algoritmo para produzir a TDE de uma imagem k-dimensional usando k transformações, uma para cada direção de coordenada. Para imagens 2D, primeiramente, os valores de distância são calculados ao longo de cada linha (*1^a transformação*), como descrito na Seção 2.4.4. Em seguida, estes valores são utilizados para calcular as distâncias mínimas ao longo de cada coluna (*2^a transformação*). Mantendo-se a notação da Seção 2.4.4, tem-se, formalmente:

Transformação 2 *A partir de G (da Transformação 1), a 2^a transformação de Saito gera uma imagem S (que pode ser construída sobre o mesmo espaço que G), o mapa de distâncias euclidianas de F, dado por:*

$$S(i,j) = \min_x \left\{ G(x,j) + (i-x)^2 \right\} \tag{2.3}$$

Note-se que a distância euclidiana ao quadrado entre dois pixels é definida pela distância ao quadrado na vertical mais a distância ao quadrado na horizontal. Após a *transformação 1*, todo pixel (x,j) da coluna j tem valor $G(x,j)$, que representa a distância na horizontal entre (x,j) e

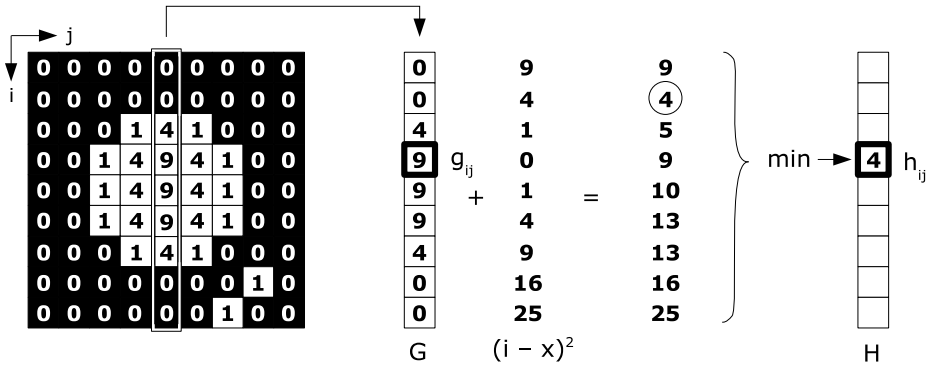


Figura 2.4: Exemplo da transformação 2 para um pixel (i, j).

o pixel preto mais próximo na linha x. Adicionando-se a $G(x, j)$ a distância vertical entre (i, j) e (x, j), $(i - x)^2$, encontra-se a distância 2D entre (i, j) e o pixel mais próximo a (x, j) na linha x. Tomando-se o mínimo dos resultados para todas as linhas x, tem-se que $S(i, j)$ será a distância entre (i, j) ao *site* mais próximo.

Saito e Toriwaki implementam a *transformação 2* usando uma varredura para baixo seguida de uma varredura para cima em cada coluna de G. Durante a varredura para baixo, para cada pixel (i, j) aplica-se um teste para restringir o número de pixels à frente na coluna j para os quais a minimização é realizada. A varredura de baixo para cima procede de similarmente. Mais detalhes desse teste pode ser encontrado no artigo original de Saito [7].

2.7 O Algoritmo de Maurer

O artigo original de Maurer et. al. [8] é uma extensão do método de Breu [47], tornando-o geral para qualquer dimensão e métrica. Espera-se apresentar aqui uma descrição mais conceitual, já que não é necessário expor a teoria geral em detalhes.

A TDE de Maurer pode ser resumida nos seguintes passos:

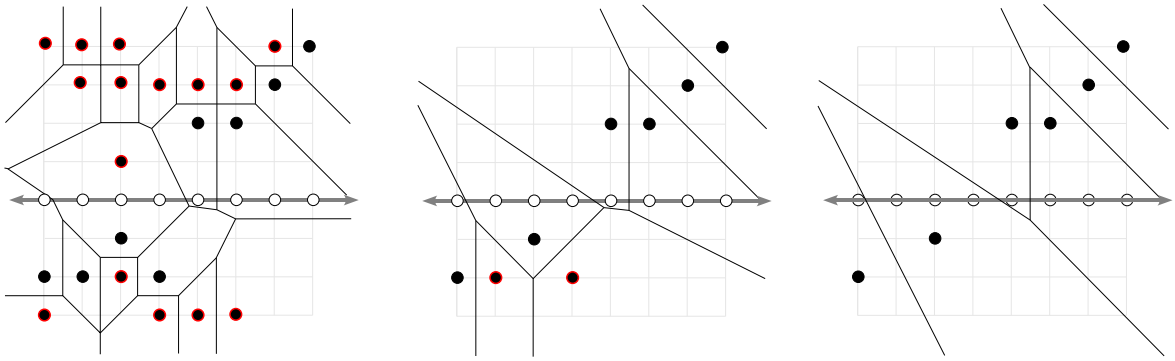
Algoritmo 2 Descrição alto-nível da TDE de Maurer

1. Calcular a TDE 1D coluna por coluna
2. Para cada linha j
   (a) Determinar a interseção do DV com a linha j.
      i. Restringir-se aos *sites* relevantes utilizando a informação da TDE 1D.
   (b) Varrer a linha j calculando a TDE através da consulta da região de Voronoi de cada pixel.

O primeiro passo é o mesmo que nos outros algoritmos de varredura independente visto na Seção 2.4.4. Porém, para facilitar a explicação, convém realizar as TDEs 1D verticalmente em vez de horizontalmente.

Com é típico dos algoritmos de varredura independente, o processamento não-trivial reside no segundo passo. O fato chave do método de Maurer é que a interseção do diagrama de Voronoi com uma linha da imagem pode ser eficientemente calculada, representada e consultada. Uma razão importante para isto é que:

- **Poucos *sites* influenciam o mapa de distâncias em uma linha da imagem**.



(a) **(b)** **(c)**

Figura 2.5: Em (a), têm-se o DV de todos os *sites*, representados por pontos pretos. *Sites* circulados em vermelho são irrelevantes para calcular a TDE na linha cinza, pois suas RVs não a interceptam. Os *sites* marcados em (a) são removidos para gerar (b), mantendo-se apenas os *sites* mais próximos em cada coluna. Em (b) removem-se os *sites* marcados, que não passam em um teste de interseção de bissetrizes. Em (c), tem-se o DV dos *sites* relevantes. A interseção do DV total (a) não se altera nos DVs parciais (b) e (c).

A Figura 2.5 ilustra este fato. A determinação de *quais sites* são relevantes para uma linha $\mathcal{R}$ envolve duas restrições:

1. Desconsiderar qualquer *site* que não seja o mais próximo de algum pixel em $\mathcal{R}$ ao longo de cada coluna. Esta informação é fornecida pela TDE 1D. Na Figura 2.5(a), os *sites* circulados em vermelho foram removidos por este critério, gerando o DV (b).

2. Sejam u, v e w três dos *sites* restantes tal que $u_x < v_x < w_x$. Seja $\widehat{uv}$ a interseção da mediatriz entre u e v com a linha $\mathcal{R}$, e seja $\widehat{vw}$ definida de maneira análoga, como mostra a Figura 2.6. Não é difícil enxergar que a região de Voronoi do *site* v não intercepta a linha $\mathcal{R}$ se $\widehat{uv}$ está à direita de $\widehat{uv}$, ou seja, $\widehat{uv}_x \geq \widehat{vw}_x$. Na Figura 2.5(b), estão marcados os *sites* que foram eliminados por esta propriedade; o DV dos sites restantes está mostrado na Figura 2.5(c).

Para a linha $\mathcal{R}$ nada mudou no DV com todos os *sites*, Figura 2.5(a), ou no DV parcial da Figura 2.5(c). Uma vez encontrados os *sites* relevantes para $\mathcal{R}$ usando as restrições acima, isto é, uma vez determinados os *sites* cujas RVs interceptam $\mathcal{R}$, fica fácil determinar a TDE final para os pixels da linha $\mathcal{R}$. Os *sites* relevantes podem ser representados apenas pela ordem em que

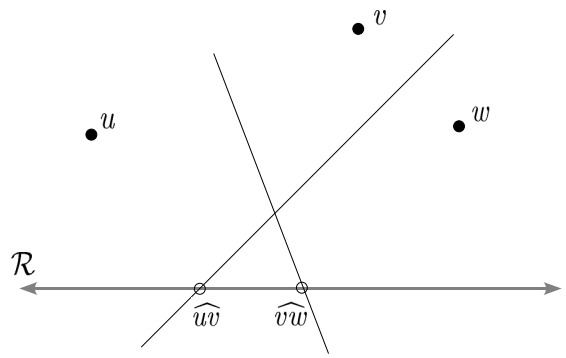


Figura 2.6: A região de Voronoi de v não intercepta a linha $\mathcal{R}$ se $\widehat{uv}_x \geq \widehat{vw}_x$.

aparecem da esquerda para a direita: $s_1, s_2, \ldots, s_m$, com $m < n_{cols}$ e $s_1.x < s_2.x < \cdots < s_m.x$. Tais *sites* ordenados também representam implicitamente a interseção do DV com a linha $\mathcal{R}$; como s_i está à esquerda de $s_i + 1$, então $RV(s_i)$ restrita a $\mathcal{R}$ está à direita de $RV(s_i + 1)$ restrita a $\mathcal{R}$, para $i = 1 \ldots m$.

Dado este conjunto de *sites* relevantes ordenados pela coluna, a TDE na linha $\mathcal{R}$ é gerada em $O(n_{cols})$ através do seguinte Algoritmo 3 para consulta ao DV.

Algoritmo 3 TDE 2D de Maurer para uma linha da imagem, dados os *sites* relevantes

1. O *site* atual s_i é igual s_1 (*site* mais à esquerda), ou seja, $i = 1$.
2. Para cada pixel p da linha $\mathcal{R}$, da esquerda para a direita, faça:
   - se $i < m$, e $d(p, s_i) > d(p, s_{i+1})$ faça $i = i + 1$
   - TDE(p) $= d(p, s_i)$

Executando-se o Algoritmo 3 para cada linha, fica claro que a complexidade do segundo passo de Maurer é $O(n^2)$. Juntamente com o primeiro passo, que também é $O(n^2)$, tem-se que a complexidade total do algoritmo de Maurer é $O(n^2)$.

A TDE 1D é utilizada tanto para eliminar *sites* não relevantes como para calcular $d(p, s_i)$ no Algoritmo 3 da seguinte forma: $d(p, s_i) = TDE_{1D}(p) + (p_x - i)^2$ (distância euclidiana ao quadrado). Trata-se do mesmo princípio utilizado pela Transformação 2 do algoritmo de Saito, descrito na Seção 2.6.

2.8 O Algoritmo de Lotufo-Zampirolli

Nesta seção, será descrito o método de Lotufo-Zampirolli, pesquisadores da UNICAMP, para calcular a TDE exata. Será assumida uma familiaridade prévia com conceitos de morfologia matemática em escala de cinza [81].

Shih e Mitchell [75] mostraram que a TDE pode ser calculada através de erosão em níveis

de cinza da imagem pela seguinte função estruturante:

$$b_e(x) = -d_e^2(x, \mathcal{O})$$

onde $\mathcal{O}$ é a origem e $d_e^2(p, q)$ é a distância euclidiana elevada ao quadrado. A imagem de entrada deve possuir valor "infinito" onde o valor das distâncias deve ser calculado (i.e. pixels brancos). Pode-se tomar como "infinito" um número qualquer maior que o máximo valor de distância possível na imagem, por exemplo o comprimento da diagonal mais um.

Esse resultado permite que algoritmos rápidos utilizados em morfologia possam ser utilizados para calcular a TDE exata. Duas técnicas comumente utilizadas para otimizar os algoritmos de morfologia são: a decomposição do elemento estruturante e a erosão por propagação. Essas técnicas combinadas formam a essência da TDE de Lotufo-Zampirolli.

Shin e Mitchell mostraram que o elemento estruturante b_e pode ser decomposto da seguinte forma:

$$b_i = \begin{bmatrix} -4i+2 & -2i+1 & -4i+2 \\ -2i+1 & \mathbf{0} & -2i+1 \\ -4i+2 & -2i+1 & -4i+2 \end{bmatrix}$$
$$b_e = b_1 \oplus b_2 \oplus b_3 \cdots$$

onde o centro de b_i está marcado em negrito e '$\oplus$' é a soma de Minkowski [81]. Dessa forma, por uma propriedade de morfologia matemática, pode-se calcular a TDE por erosões sucessivas:

$$\varepsilon_{b_e}(f) = \cdots \varepsilon_{b_3}(\varepsilon_{b_2}(\varepsilon_{b_1}(f)))$$

No método de Lotufo-Zampirolli, o elemento estruturante é decomposto ainda mais, resultando em elementos 1D nas direções norte (N), sul (S), leste (E), e oeste (W):

$$b_{N_i} = \begin{bmatrix} -2i+1 \\ 0 \end{bmatrix}, \; b_{E_i} = \begin{bmatrix} 0 & -2i+1 \end{bmatrix},$$
$$b_{S_i} = \begin{bmatrix} 0 \\ -2i+1 \end{bmatrix}, \; b_{W_i} = \begin{bmatrix} -2i+1 & 0 \end{bmatrix}$$
$$\begin{aligned} b_e = &\cdots \oplus b_{N_2} \oplus b_{N_1} \oplus \cdots \oplus b_{S_2} \oplus b_{S_1} \oplus \cdots \\ &\cdots \oplus b_{W_2} \oplus b_{W_1} \oplus \cdots \oplus b_{E_2} \oplus b_{E_1}. \end{aligned} \tag{2.4}$$

Por esse motivo, e pela propriedade de idempotência de morfologia [81], a transformada de distância pode ser calculada erodindo-se cada coluna da imagem por $b_{N_1}, b_{N_2}, \ldots$ até ocorrer estabilidade (a coluna não muda mais), e, igualmente, erodindo-se as colunas por $b_{S_1}, b_{S_2}, \ldots$, seguido da erosão das linhas por $b_{E_1}, b_{E_2}, \ldots$ e, finalmente, erodindo-se as linhas por $b_{E_1}, b_{E_2}, \ldots$. O algoritmo utiliza-se de filas de pixels para processar somente aqueles que mudam de erosão em erosão. Maiores detalhes podem ser obtidos no artigo do método [4].

2.9 Propagação de Vizinhanças Múltiplas de Cuisenaire

O método de TDE proposto por O. Cuisenaire e B. Macq em 1999 [6, 43] consiste basicamente em adicionar uma etapa de correção ao algoritmo PSN descrito na Seção 2.4.1. Como visto na Seção 2.3, a TDE exata pode ser gerada pelo PSN com vizinhança suficientemente grande. A propagação com uma vizinhança grande, no entanto, é muito custosa. Sendo assim, Cuisenaire propõe que, após a PSN com vizinhança-de-4, a propagação com vizinhanças maiores seja realizada apenas em poucos locais necessários previstos em teoria. Como explicado a seguir, tais locais são alguns pontos em torno das fronteiras das regiões de Voronoi que possam levar a uma desconexão da mesma, causando erro.

2.9.1 Propagação por vizinhança fixa

O algoritmo PSN proposto por Cuisenaire possui algumas particularidades em relação à descrição dada na Seção 2.4.1. Cada nó do conjunto de contorno armazena um ponto $p = (p_x, p_y)$ e a distância relativa $dp = (dp_x, dp_y)$ da fonte mais próxima a p. O vetor dp é utilizado para determinar a distância do vizinho do pixel corrente p à fonte mais próxima a p da seguinte forma. Para $n \in \mathcal{N}$, o vizinho de p é dado por $q = p + n$, e sua distância à fonte mais próxima de p é $\|dp + n\|^2 = (dp_x + n_x)^2 + (dp_y + n_y)^2$. O Agoritmo 4 descreve o PSN de Cuisenaire com esses detalhes. Para uma melhor compreensão, o leitor deve compará-lo ao procedimento geral de propagação (Algoritmo 1).

Algoritmo 4 PSN de Cuisenaire

1. Todos os pixels brancos recebem distância ∞.
2. Insira em $bucket(0)$ os pixels brancos e coordenadas relativas $(0, 0)$.
3. $d = 0$ {*Distância atual inicia em 0.*}
4. Enquanto o conjunto de contorno (*buckets*) não estiver vazio

   Enquanto $bucket(d)$ não estiver vazio

   Remova p e dp do $bucket(d)$ {*pixel de menor custo.*}

   Para todo $n \in \mathcal{N}$

   $D_{nova} = \|dp + n\|^2$

   Se $D_{nova} < D(p + n)$

   {p *propaga para* $p + n$.}

   $D(p + n) = D_{nova}$

   Insira $(p + n, dp + n)$ em $bucket(D_{nova})$.

   fim se

   d = d + 1

Na prática, a etapa 2 inicializa o conjunto de contorno com os pixels de fronteira, ou seja, os pixels brancos com algum vizinho-de-4 preto. Tais pixels estão a distância 1 do conjunto de interesse e, portanto, são inseridos em $bucket(1)$, e a distância atual do passo 3 começa em 1. Dessa forma, a propagação trivial dos pixels com distância zero para os pixels a distância unitária

é realizada diretamente na etapa de inicialização, poupando um ciclo de inserção e propagação.

É possível determinar se todos os *buckets* estão vazios no passo 4 do Algoritmo 4 se apenas uma determinada quantidade dos últimos *buckets* até o *bucket* atual d estiverem vazios [43]. Para a vizinhança de 4, basta testar se os últimos $2\sqrt{d}+1$ *buckets* estão vazios, pois $\|dp+n\|^2 \leq (\sqrt{d}+1)^2 = d+2\sqrt{d}+1$.

2.9.2 Propagação por vizinhanças múltiplas

Durante a PSN, o método verifica se um pixel p propagou informação de distância mínima para algum vizinho. Se p estiver proximo da fronteira de uma RV $\mathcal{N}$-conectada, então ele não será propagado para nenhum vizinho, pois eles estarão mais perto de outros *sites* e, portanto, estão em RVs diferentes da de p. Como justificado mais adiante, apenas nesses pixels não-propagantes uma propagação com vizinhança maior pode ser necessária.

Por exemplo, regiões de Voronoi desconectadas causam erros na propagação simples com vizinhança de 4, como discutido na Seção 2.3 e ilustrado na Figura 2.1(a). Para corrigir este tipo de erro, pode-se usar uma vizinhança maior na fronteira da RV 4-conectada para recuperar a conectividade. Na Figura 2.1(a), é suficiente propagar o pixel entre p_2 e q com uma vizinhança-de-8 para corrigir o mapa. No entanto, para outras imagens um erro do tipo da Figura 2.1(c) pode ocorrer. Novamente, a idéia é utilizar uma vizinhança ainda maior, por exemplo 5×5, no pixel marcado com distância 116 na Figura 2.1(c). O método de Cuisenaire é uma realização eficiente desta idéia [6, 43].

Como descrito na Seção 2.3, a TDE para uma vizinhança $\mathcal{N}$ ($\mathcal{N}$-TDE) é correta para toda distância menor que um determinado valor $d_c(\mathcal{N})$. Além disso, o valor de distância $d_{np}(p)$ de um pixel não-propagante p que leva a um erro é bem definida para uma vizinhança $\mathcal{N}$.

Propriedade 6 *Para cada vizinhança $\mathcal{N}$, existe uma distância $d_{np}(\mathcal{N})$ tal que:*
$D(p) \geq d_{np}(\mathcal{N}) \iff$ para alguma imagem, p não é propagado durante a $\mathcal{N}$-TDE e gerou um erro de desconexão em sua região de Voronoi.

Em termos mais simples, $d_{np}(\mathcal{N})$ é o mínimo valor de distância em que pode ocorrer um pixel não-propagante gerador de erro na $\mathcal{N}$-EDT. Logo, nem todos os pixels não-propagantes em uma $\mathcal{N}$-EDT necessitam de uma propagação maior – apenas aqueles com uma distância maior ou igual a $d_{np}(\mathcal{N})$. A propagação múltipla pode parar para todos os pixels com distância suficientemente pequena.

O valor de d_{np} para uma vizinhança $\mathcal{N}$ pode ser encontrado por um algoritmo de busca exaustiva, da mesma forma que d_c é calculado (vide Seção 2.3). O algoritmo para $d_{np}(\mathcal{N})$ foi apenas descrito vagamente por Cuisenaire [6, 43]. A implementação em linguagem C foi realizada pelo autor desta dissertação e encontrar-se-á na biblioteca de rotinas Animal [49]. A Tabela 2.1 mostra valores de d_c e d_{np} para diversas vizinhanças. As posições relativas onde pode ocorrer um erro e do pixel não-propagante correspondente também estão listadas na tabela. Ademais, define-se $\mathcal{N}_1$ como a vizinhança de 4, e $\mathcal{N}_k$ é a vizinhança quadrada $2k-1 \times 2k-1$ para $k > 1$, como nas primeiras duas colunas da Tabela 2.1.

Vizinhança		**Erro mais próximo**		**Não-propagante mais próximo**	
k	$\mathcal{N}_{\mathbf{k}}$	posição relativa	d_c	posição relativa	d_{np}
1	4-vizinhos	(2,2)	8	(1,1)	2
2	3×3	(12,5)	169	(10,4)	116
3	5×5	(28,8)	848	(25,7)	674
4	7×7	(48,10)	2404	(44,9)	2017
5	9×9	(72,12)	5328	(67,11)	4610
6	11×11	(108,15)	11889	(102,14)	10600
7	13×13	(143,17)	20738	(136,16)	18752
8	15×15	(192,20)	37264	(184,19)	34217
9	17×17	(238,22)	57128	(229,21)	52882
10	19×19	(300,25)	90525	(290,24)	84676
11	21×21	(357,27)	128178	(346,26)	120392
12	23×23	(420,29)	177241	(408,28)	167248
13	25×25	(500,32)	251024	(487,31)	238130
14	27×27	(574,34)	330632	(560,33)	314689
15	29×29	(667,37)	446258	(652,36)	426400
16	31×31	(768,40)	591424	(752,39)	567025
17	33×33	(858,42)	737928	(842,41)	708962
18	35×35	(972,45)	946809	(954,44)	912052
19	37×37	(1054,46)	1113032	(1035,45)	1073250
20	39×39	(1200,50)	1442500	(1180,49)	1394801
21	41×41	(1312,52)	1724048	(1291,51)	1669282
22	43×43	(1452,55)	2111329	(1430,54)	2047816
23	45×45	(1575,57)	2483874	(1552,56)	2411840
24	47×47	(1680,58)	2825764	(1656,57)	2745585
25	49×49	(1862,62)	3470888	(1837,61)	3378290

Tabela 2.1: Erros e posições não-propagantes mais próximos para diversas vizinhanças.

Os valores da Tabela 2.1 foram gerados pela implementação da busca exaustiva realizada pelo autor desta monografia. Os valores gerados foram idênticos àqueles publicados por Cuisenaire em seu artigo [6] para todas as linhas até $k = 16$, exceto pela linha 3, isto é, vizinhança 5×5. Logo, a linha 3 provavelmente é um erro na tabela publicada por Cuisenaire. Para $k > 16$, as linhas nesta tabela não haviam sido publicadas antes desta dissertação.

O algoritmo PMN de Cuisenaire consiste em propagar sucessivamente com vizinhanças crescentes $\mathcal{N}_k$ onde necessário. Dado um pixel p não-propagante, que permaneceu na fila de prioridades após a $\mathcal{N}_1$-TDE, p será propagado pela menor vizinhança necessária para corrigir quaisquer erros relacionados. Por exemplo, se $D_{\mathcal{N}_1}(p) = 3000$, da Tabela 2.1 verifica-se que p deverá ser propagado novamente por uma vizinhança 9×9 para garantir que ele não gere erro, pois 3000 está entre 2017 e 4610. Mais especificamente, de todos os pixels p não-propagantes em uma $\mathcal{N}_k$-TDE, os que são propagados com $\mathcal{N}_{k+1}$ são aqueles que satisfazem $d_{np}(\mathcal{N}_k) \leq D_{\mathcal{N}_k}(p) < d_{np}(\mathcal{N}_{k+1})$. O Algoritmo 5 descreve em detalhes a TDE por propagação com múltiplas vizinhanças.

Algoritmo 5 Propagação de Vizinhanças Múltiplas de Cuisenaire (PMN)

Entrada:

$\mathcal{N}_1$-TDE: saída do Algoritmo 4;

buckets: contendo os pixels que não propagaram durante a PSN;

d_{max}: máxima distância usada na PSN;

k_{max}: Número de vizinhanças para as quais $d_{np}(\mathcal{N}_k)$ foi pré-calculado.

Saída:

D: $\mathcal{N}_k$-TDE com $k = k_{max}$. Esta TDE é exata se a imagem conter distâncias menores que $d_c(\mathcal{N}_k)$, mas pode conter erros caso contrário.

Explicação de variáveis:

d: distância corrente; indexa o vetor de *buckets*.

k: indexa vizinhança, ou seja, indica a vizinhança $\mathcal{N}_k$

início

Para $k = 2$ até k_{max}

Para $d = d_{np}(\mathcal{N}_{k-1})$ até o mínimo entre $d_{np}(\mathcal{N}_k)$ e d_{max}

Para todo $n \in \mathcal{N}_k$

$D_{nova} = \|dp + n\|^2$

Se $D_{nova} < D(p + n)$

$D(p + n) = D_{nova}$

Insira $(p + n, dp + n)$ em $bucket(D_{nova})$.

fim se

fim

Cuisenaire afirma que seu algoritmo parece permanecer $O(n^2)$ mesmo no pior caso, baseado em testes empíricos [6, 43]. Entretanto, esta conjectura ainda não foi demonstrada.

Deve-se notar que a etapa de se pré-calcular as distâncias $d_{np}(\mathcal{N}_i)$ para um i grande é bastante custosa. Porém, uma vez calculados tais valores, eles são disponíveis para sempre, sem

precisarem ser re-calculados.

A propagação pode ser restrita apenas às direções necessárias, uma variante do algoritmo denominada PMON [43]. Ainda não está claro se esta variante é mais eficiente que o PMN, por causa do custo adicional de se calcularem as direções de propagação. Alguns testes empíricos de Cuisenaire sugerem que o PMON é mais rápido que PMN para imagens suficientemente grandes.

2.10 Image Foresting Transform

2.10.1 Introdução à IFT

A *Image Foresting Transform* (IFT) é uma abordagem unificada e eficiente para técnicas de processamento de imagens como crescimento de regiões, transformada *watershed*, transformada de distâncias, esqueletos, dentre várias outras [52]. A seguir, serão explicados gradualmente os elementos que compõem a abstração da IFT.

A IFT concebe uma imagem digital como um *grafo*, em que os pixels são os *nós* do grafo e os *arcos* são definidos de acordo com uma *relação de adjacência* (vizinhança) entre pixels. Por exemplo, uma possível relação de adjacência $\mathcal{N}$ define que existe um arco entre dois pixels p e q se a distância euclidiana entre eles é menor ou igual a um dado raio r. Formalmente:

$$\mathcal{N}(p) = \{(p, q) \mid d(p, q) \leq r\}$$

Para $r = 1$, tem-se a vizinhança de 4 usual. Para $r = \sqrt{2}$, tem-se a vizinhança de 8.

Uma *função peso* associa um número não-negativo a cada arco, representando seu "comprimento" (em um sentido amplo). Esse "comprimento" pode levar em conta diversos fatores, dependendo da aplicação. É comum adotar o comprimento do arco (p, q) como a distância euclidiana entre p e q. Em outras aplicações, esse comprimento pode ser, por exemplo, o módulo da diferença de intensidade entre p e q.

São especificadas n *sementes*. Cada semente é um conjunto de pixels, e um número é atribuído a cada semente. Esse número é denominado *rótulo* ou *etiqueta* da semente. A "distância" de uma semente a um pixel p é definida como o "comprimento" do menor caminho que liga a semente ao pixel p na imagem (vista como um grafo). Para medir o comprimento de um caminho, utiliza-se uma função que leva em conta os pesos dos arcos que compõem o caminho. Essa função, chamada *função custo de caminho*, pode ser, por exemplo, a soma dos pesos dos arcos ao longo do caminho ou, alternativamente, o máximo desses pesos ou o produto dos mesmos.

A IFT tem como objetivo encontrar a região de influência de cada semente pré-definida. Na linguagem de teoria dos grafos, a região de influência de uma semente é uma árvore de menor caminho. Em uma árvore desse tipo, a raíz é a semente e cada nó é um pixel. O caminho dessa árvore que liga um pixel até a semente é sempre um menor caminho, ou seja, todos os outros caminhos ligando esse pixel até a semente possuem um comprimento igual ou maior a este. Deve-se lembrar que "comprimento" é sempre medido com a função custo de caminho adotada.

Como são definidas várias sementes, a IFT encontra várias árvores de menor caminho, uma para cada semente. Diz-se que essas várias árvores formam uma "floresta". Daí o nome da técnica, que, traduzido, fica "transformada de *florestamento* de imagens".

Resumindo, para cada problema a ser resolvido pela IFT, devem ser definidos: n sementes e seus rótulos, uma relação de adjacência, uma função peso e uma função custo de caminho. Em um algoritmo de crescimento de regiões, por exemplo, pode-se usar a relação de adjacência igual à vizinhança de 4, a função peso como sendo o módulo da diferença do brilho de pixels adjacentes, e uma função custo de caminho que associa a cada caminho o peso de seu arco mais pesado. As sementes podem ser definidas pelo usuário. A IFT vai, então, definir as árvores de menor caminho que ligam cada pixel da imagem à sua semente mais próxima. No caso do crescimento de regiões, as árvores de menor caminho irão agrupar os pixels mais similares às sementes definidas pelo usuário, resultando uma segmentação da imagem.

Em mais detalhes, o que a IFT faz é crescer uma árvore de menor caminho, simultaneamente, de cada semente no grafo, propagando as seguintes propriedades para cada pixel p em uma imagem de anotação:

- A etiqueta da semente mais próxima de p;
- A distância à semente mais próxima de p. Essa distância é o comprimento do menor caminho que liga p à sua semente mais próxima;
- O pixel adjacente a p, denominado seu pixel-pai, que o leva de volta à sua semente mais próxima através de um menor-caminho.

Ao menos uma dessas três propriedades deve ser relevante para um dado problema. Por exemplo, a segmentação por crescimento de regiões é dada pela imagem com as etiquetas propagadas, e transformadas de distância são obtidas na imagem das distâncias. O pixel-pai é utilizado para o cômputo de caminhos geodésicos, por exemplo em bordas.

Para ilustrar o conceito de particionamento da imagem gerado pela IFT, seja o grafo da Figura 2.7(a) contendo duas sementes representadas por nós coloridos. Os rótulos são representados pelas cores das sementes. Ademais, os pesos das arestas são representados por números próximos a elas e a relação de adjacência é a vizinhança de 4. A IFT irá crescer uma árvore de caminho mínimo a partir de cada semente, até que ambas as árvores cubram a imagem toda (o processo detalhado será dado adiante). Neste exemplo, o custo de um caminho é medido multiplicando-se os pesos ao longo do caminho. O resultado do particionamento – uma floresta de caminhos mínimos – está ilustrado na Figura 2.7(b). Tem-se duas árvores representando a região de influência de cada semente. O caminho que liga um nó à sua semente na sua árvore é um garantidamente um caminho mínimo.

Na Figura 2.7(b), os rótulos propagados pela IFT são representados por cores iguais às das etiquetas das sementes. Além disso, a informação de pixel-pai está representada por uma flecha apontando para o próximo pixel ao longo de um menor caminho. Dado um pixel qualquer, basta seguir a flecha para percorrer o menor caminho até a semente mais próxima. Deve-se notar que

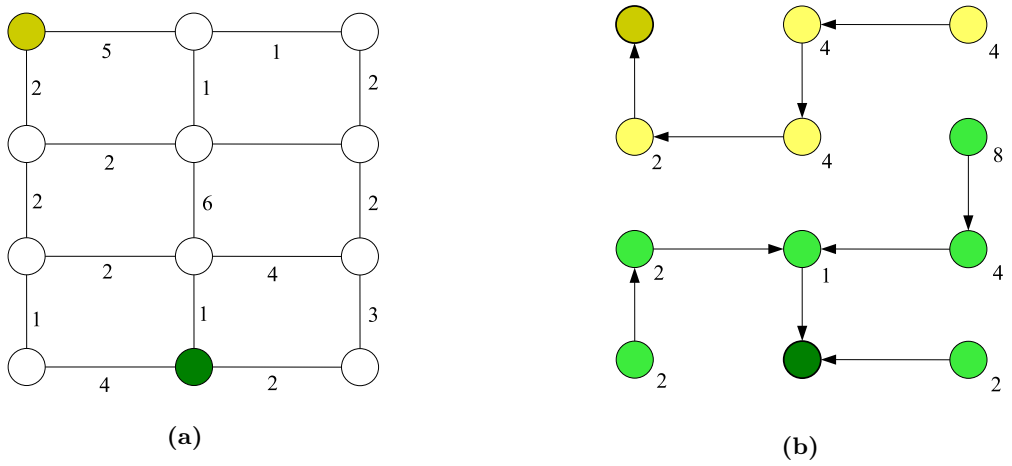


Figura 2.7: Exemplo de uma partição de um grafo em uma floresta de caminhos mínimos a partir de duas sementes (nós coloridos em (a)). Os custos das arestas estão representados próximos às mesmas, em (a). A função custo de caminho utilizada é a multiplicação dos pesos do caminho. Em (b), tem-se representadas as zonas de influência de cada semente, dadas pelas árvores, com os custos mínimos aparecendo próximos aos vértices.

o intuito deste exemplo simples é apenas ilustrar a idéia de floresta de caminhos mínimos, não sendo ligado a nenhuma aplicação específica.

O Algoritmo 6 mostra a IFT abstrata, sobre a qual se baseia o algoritmo de transformada de distância a ser descrito. Note-se a semelhança do algoritmo da IFT com o processo geral do Algoritmo 1 visto na Seção 2.4.1. Como já foi dito, o ponto chave desse tipo de algoritmo consiste na implementação da fila de prioridades Q. Com essa fila devidamente implementada [58, 59, 52], o algoritmo é tempo-linear no número de pixels, ou seja, $O(n^2)$ para uma imagem $n \times n$. Na biblioteca Animal (An IMAging Library) [49], estão os arquivos `ift_pqueue.h` e `ift_pqueue.c`, que contêm a implementação da fila de prioridades realizada pelo autor desta dissertação como sugerido pelo prof. Alexandre Falcão, autor do método.

2.10.2 TDE por IFT

A transformada de distância euclidiana pode ser calculada eficientemente através da IFT, com boa precisão. Para isto, deve-se formular o problema da transformada da distância sob os conceitos da IFT:

- **Nós do grafo:** pixels da imagem;
- **Relação de adjacência:** vizinhança de 8. Entretanto, qualquer outra vizinhança simétrica pode ser utilizada. Para vizinhanças maiores, o custo do algoritmo aumenta e o ganho em precisão é pouco.
- **Sementes:** pixels '0' (não-objeto). Pode-se primeiro detectar a borda binária do objeto e tomar seus pixels como as sementes;

Algoritmo 6 IFT geral

Entrada: uma imagem I, uma relação de adjacência $\mathcal{N}$, um conjunto S de sementes rotuladas e uma função custo de caminho $pf(\mathcal{C}_p)$, que calcula o custo de se percorrer o caminho $\mathcal{C}_p$ de uma semente até o pixel p.
Saída: três imagens, *custo* (custos acumulados dos caminhos até cada pixel, em um dado instante), *pai* (pixel pai) e *rot* (rótulos), que representam a floresta computada.
Estruturas de dados auxiliares: uma fila de prioridades Q, uma lista L de pixels que já terminaram de ser processados e uma variável auxiliar *tmp*.

início

1. para todo pixel $p \in I$, atribua ∞ para $custo[p]$ e *nil* para $rot[p]$ e $pai[p]$;
2. para todo pixel semente $s \in S$, atribua 0 para $custo[s]$, o rótulo associado a s para $rot[s]$, e insira s em Q;
3. *enquanto* Q não estiver vazia *faça*
   (a) remova o pixel p de Q tal que $custo[p] = min_{p' \in Q}\{custo[p']\}$ e insira p em L;
   (b) *para* cada pixel q adjacente a p de acordo com $p, q \notin L$, *faça*
      i. calcule $tmp \leftarrow pf(\mathcal{C}_p \cdot \langle (p,q) \rangle)$, representando o custo do caminho para se chegar a q passando por p. (a notação $\mathcal{C}_p \cdot \langle (p,q) \rangle$ significa "concatenação do caminho $\mathcal{C}_p$ com o arco (p,q)".)
      ii. se $tmp < custo[q]$ então
         A. atribua tmp para $custo[q]$, $rot[p]$ para $rot[q]$ e p para $pai[q]$;
         B. *se* $q \notin Q$ *então* insira q em Q, *senão* atualize a posição de q em Q;
         fim se;
      fim para;
   fim enquanto;

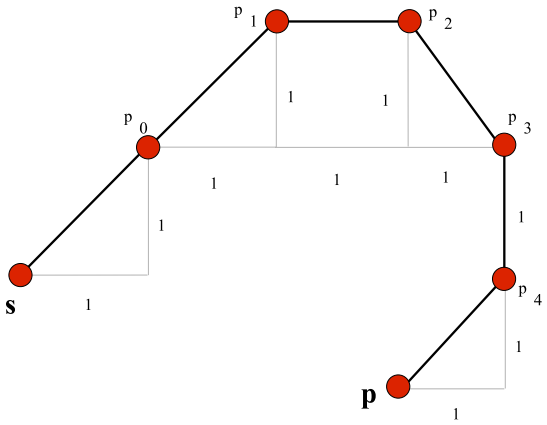
fim

- **Pesos das arestas:** implícitos na função custo de caminho a ser definida no próximo item;
- **Função custo de caminho:** pode ser definida como:

$$pf(\mathcal{C}) = \Big(\sum_{i=1}^{n-1} |x_{p_i} - x_{p_i+1}|\Big)^2 + \Big(\sum_{i=1}^{n-1} |y_{p_i} - y_{p_i+1}|\Big)^2 \tag{2.5}$$

  A raíz quadrada da distância euclidiana não é utilizada, pois assim apenas números inteiros são manipulados. Podemos imaginar que uma aresta (p,q) tenha custo dado pela diferença $pf(s, \cdots, p, q) - pf(s, \cdots, p)$. Vale notar que o custo do caminho calculado pela Equação 2.5 não depende somente da posição do primeiro e do último pixel do caminho, como seria natural de se pensar, uma vez que está sendo usada a soma dos valores absolutos dos deslocamentos.

Na forma em que a Equação 2.5 está colocada, o custo requerido para computar esta função é muito alto, uma vez que se tem que calcular duas somatórias para cada aresta sendo analisada. Pode-se diminuir esse custo armazenando para cada pixel os vetores de deslocamentos (dx, dy) ao longo do caminho da semente mais próxima até esse pixel, como mostrado na Figura 2.8. Nessa figura, quando há um deslocamento horizontal, incrementa-se o contador $dx[p]$, e quando há um deslocamento vertical, incrementa-se o contador $dy[p]$. Dessa maneira, o custo acumulado do caminho mais curto a um pixel p encontrado até um determinado instante pode ser calculado



$$dx[p] = 1 + 1 + 1 + 1 + 1 = 5$$
$$dy[p] = 1 + 1 + 1 + 1 + 1 = 5$$
$$pf(\langle s, p_0, p_1, p_2, p_3, p_4, p\rangle) = (dx[p])^2 + (dy[p])^2 = 50$$

Figura 2.8: Cálculo do custo de um dado caminho ligando dois pixels s e p, utilizado no cálculo da TDE por IFT. Quando há um deslocamento horizontal, incrementa-se o contador $dx[p]$, e quando há um deslocamento vertical, incrementa-se o contador $dy[p]$. Desta maneira, evita-se recalcular as somatórias presentes na Equação 2.5. Esse custo mede o comprimento de arco do caminho como se estivesse "esticado".

como $(dx[p])^2 + (dy[p])^2$, não sendo mais preciso calcular somatórias a cada nova avaliação de aresta.

Com os elementos da IFT definidos como supracitado, o Algoritmo 6 pode ser utilizado para calcular a TDE com boa precisão. Entretanto, o algoritmo pode ser mais especializado de modo a considerar os vetores de deslocamento dx e dy para o cômputo da função custo de caminho, gerando o Algoritmo 7.

Terminado o algoritmo, o mapa de distâncias (ao quadrado) se encontra na imagem *custo*. As imagens de saída *pai* e *rot* não são necessárias à TDE mas são bastante úteis para o cálculo de entidades geométricas como esqueletos (diagramas de Voronoi pontuais), como mostra o artigo [82].

Como já foi dito, este algoritmo é $O(n^2)$ para uma imagem $n \times n$. A TDE por IFT foi implementada pelo autor desta monografia, cujo código foi disponibilizado na biblioteca Animal, sob licença GPL [49]. O leitor interessado deverá procurar pelo código na função `distance_transform` nos arquivos `analysis.c` e `analysis.h`, juntamente com a rotina `euclidean_propagation` nos arquivos `ift.c` e `ift.h` da biblioteca Animal. Esta última rotina é inteiramente análoga ao Algoritmo 7.

Algoritmo 7 Transformada de distância euclidiana por IFT

Entrada: uma imagem I, uma relação de adjacência $\mathcal{N}$ (vizinhança de 8), um conjunto S de sementes rotuladas (pixels '0').
Saída: três imagens, *custo* (TDE), *pai* (pixel pai) e *rot* (rótulos), que representam a floresta computada.
Estruturas de dados auxiliares: uma fila de prioridades Q, uma lista L de pixels que já terminaram de ser processados, duas imagens dx e dy, que armazenam o deslocamento da semente mais próxima até cada pixel e uma variável auxiliar *tmp*.

início

1. para todo pixel $p \in I$, atribua ∞ para $custo[p]$, $dx[p]$ e $dy[p]$, *nil* para $rot[p]$ e $pai[p]$;
2. para todo pixel semente $s \in S$, atribua 0 para $custo[s]$, $dx[p]$ e $dy[p]$, o rótulo associado a s para $rot[s]$, e insira s em Q;
3. *enquanto* Q não estiver vazia *faça*
   (a) remova o pixel p de Q tal que $custo[p] = min_{p' \in Q}\{custo[p']\}$ e insira p em L;
   (b) *para* cada pixel q adjacente a p tal que $p, q \notin L$, *faça*
      i. calcule $tmp \leftarrow (dx[p] = |x_p - x_q|)^2 + (dy[p] = |y_p - y_q|)^2$, representando o custo do caminho para se chegar a q passando por p;
      ii. se $tmp < custo[q]$ então
         A. faça $custo[q] \leftarrow tmp$, $rot[q] \leftarrow rot[p]$, $pai[q] \leftarrow p$, $dx[q] \leftarrow dx[p] + |x_p - x_q|$ e $dy[q] \leftarrow dy[p] + |y_p - y_q|$;
         B. *se* $q \notin Q$ *então* insira q em Q, *senão* atualize a posição de q em Q;
         fim se;
   fim para;

fim enquanto;

fim

2.11 O Algoritmo de Eggers

Duas técnicas de propagação ordenada foram introduzidas por Eggers, as propagações suficientes d_∞ e d_1, que restringem a propagação a caminhos mínimos euclidianos [5]. Neste trabalho, apenas a propagação d_∞ é estudada. De acordo com Eggers, esta apresenta um melhor desempenho médio do que a propagação d_1.

Eggers separa os pixels no conjunto de contorno em duas listas, denominadas $\mathcal{L}$ e ℓ. Na primeira guardam-se os chamados *principais contorno principais*, e na segunda os *pixels de contorno secundários*. Os vizinhos de um determinado pixel são numerados como abaixo:

$$\begin{array}{ccc} N_3(p) & N_2(p) & N_1(p) \\ N_4(p) & p & N_0(p) \\ N_5(p) & N_6(p) & N_7(p) \end{array}$$

Para cada pixel de contorno associa-se um *índice de direção* k que aponta ao vizinho $N_k(p)$ para o qual p deve propagar informação. Em vez de propagarem a informação de distância para todos os oito vizinhos, os pixels de contorno principais propagam apenas para seus três vizinhos $N_k(p)$, $N_{k+1}(p)$ e $N_{k-1}(p)$, onde k é seu o índice de direção. Os pixels secundários de contorno propagam apenas para $N_k(p)$. Além disso, os índices de direção k dos pixels de contorno principais são ímpares (indicando vizinhos indiretos), enquanto os índices dos pixels secundários são pares.

O Algoritmo 8 descreve a propagação d_∞, cujas etapas gerais são as mesmas do Algoritmo 1 da seção 2.4.1. Na prática, quatro listas dinâmicas são utilizadas, duas para cada iteração. $\mathcal{L}_1$ e ℓ_1 armazenam os pixels de contorno principais e secundários, respectivamente, para a iteração corrente. Já $\mathcal{L}_2$ e ℓ_2 armazenam os pixels de contorno principais e secundários a serem utilizados na próxima iteração.

A Figura 2.9 ilustra o processo de propagação d_∞ para uma imagem binária com um único pixel de interesse. Deve-se notar que a informação de p é propagada a um pixel q através do caminho euclidiano mais curto de p a q, que consiste de $d_1(p,q) - d_\infty(p,q)$ vizinhos indiretos seguidos de $2 \cdot d_\infty(p,q) - d_1(p,q)$ vizinhos diretos. Este processo de propagação é denominado propagação d_∞ pois, iniciando-se com um único pixel preto p, o conjunto de contorno na iteração $m+1$ é a circunferência da métrica d_∞ com raio m.

Algoritmo 8 Propagação d_∞ de Eggers

1. Inicialize a distância de todo pixel branco para um valor suficientemente alto.
2. Inicialize o *Conjunto de Contorno* fazendo, para cada pixel p:
   - para $k = 0, 2, 4$ e 6, se $N_k(p)$ for branco, guarde p em $\mathcal{L}_1$ com índice de direção igual a $k + 1$.
3. Enquanto o conjunto de contorno $\mathcal{L}_1 \bigcup \ell_1$ for não-vazio:
4. para cada pixel $p \in \ell_1$
   (a) Calcule uma nova distância para o vizinho direto $N_k(p)$ baseado no valor de p.
   (b) Se a nova distância do vizinho for menor que sua distância corrente:
      - Atualize sua distância corrente.
      - Coloque $N_k(p)$ em ℓ_2 com índice de direção k.
5. Para cada pixel $p \in \mathcal{L}_1$:
   (a) Calcule uma nova distância para o vizinho indireto $N_k(p)$ baseado no valor de p.
   (b) Se essa nova distância de $N_k(p)$ for menor que sua distância atual:
      - Atualize sua distância corrente.
      - Coloque $N_k(p)$ em $\mathcal{L}_2$ com índice de direção k.
   (c) Calcule uma nova distância para os dois vizinhos diretos $N_{k+1}(p)$ e $N_{k-1}(p)$ baseado no valor de p.
   (d) Se a nova distância de $N_{k+1}(p)$ for menor que sua distância corrente:
      - Atualize sua distância corrente.
      - Coloque $N_{k+1}(p)$ em ℓ_2 com índice de direção $k + 1$.
   (e) Faça o mesmo para o item $N_{k-1}(p)$, inserindo-o com índice de direção $k - 1$ ao ser inserido em ℓ_2.
6. Faça $\mathcal{L}_1 = \ell_2$, $\mathcal{L}_2 = \emptyset$, $\ell_1 = \ell_2$, e $\ell_2 = \emptyset$.

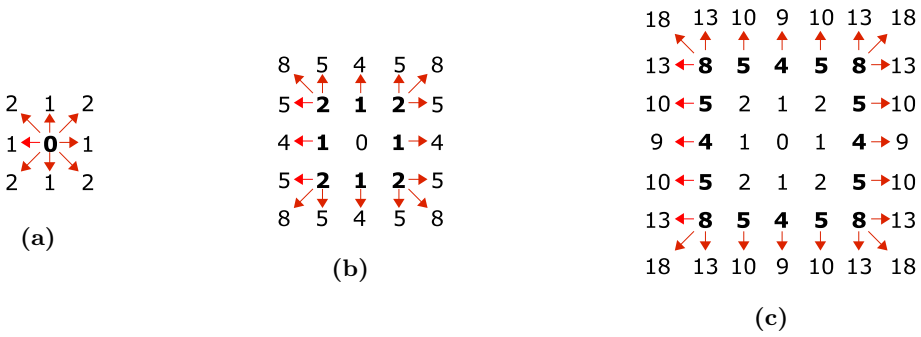


Figura 2.9: Primeiras três iterações da propagação suficiente d_∞ de Eggers. Valores em negrito indicam os pixels no conjunto de contorno. Pixels principais de contorno são os que propagam para três pixels, e os secundários são os que propagam para apenas um. Um vizinho indireto $N_k(p)$ de um pixel principal p que possui índice direcional k será um pixel principal na próxima iteração. Os vizinhos restantes serão pixels secundários.

Parte II

Avaliação Comparativa dos Algoritmos

Capítulo 3

Metodologia

Neste trabalho, foram testados e comparados os principais algoritmos recentes de TDE, que serão denominados por: Maurer [8], PMN (ou Cuisenaire) [6], Saito [7], Lotufo-Zampirolli [4] e Eggers [5]. Alguns desses algoritmos são dependentes do tamanho da imagem e do número de pixels de interesse, outros independem do número de pixels, ou dependem de algum outro fator como a distância máxima calculada (no caso de propagação), e de fatores geométricos como orientação e espessura das regiões de Voronoi.

3.1 Imagens Teste

Para estudar a velocidade e exatidão de acordo com o conteúdo e tamanho da imagem, testes com as seguintes imagens foram realizados neste trabalho:

1. **Um pixel preto ou branco em um canto**. A TDE, neste caso, produz, a maior e menor distâncias possíveis para o dado tamanho de imagem, que são $n\sqrt{2}$ (a diagonal) e 1, respectivamente. Ademais, o número de pixels que recebe uma distância não-nula é também respectivamente o maior e menor possível. Para os algoritmos de varredura, nem todas as passadas são necessárias para calcular a TDE deste tipo de imagem, dependendo do canto escolhido para o único pixel. Esta imagem-teste também foi utilizada por Maurer et. al. [8].

2. **Um círculo branco inscrito na imagem**. Este teste foi sugerido por Saito [7] e também por Cuisenaire [6]. Trata-se de um bom teste para exatidão, principalmente devido ao seguinte fato. O diagrama de Voronoi dos pixels ao longo de uma circunferência é bastante regular no plano contínuo – trata-se de diversas linhas radiais, e as regiões de influência são regiões triangulares bastante finas, cujas pontas se encontram no centro do círculo. Entretanto, as RVs discretas neste caso são irregulares, especialmente no centro do círculo. Além disso, quanto maior o círculo, mais densa é a amostragem das RVs, porém tais RVs foram mais finas.

3. **Imagem meia-preenchida**. Trata-se do pior caso do algoritmo de força-bruta, como visto na Seção 2.1. Esta imagem também possui uma distribuição uniforme de distâncias, isto é, há um número aproximadamente igual de pixels para cada valor diferente de distância na TD.

4. **Pixels pretos aleatórios, compondo 1%, 2%, 5, 10, 20,..., 90, 95, 98 e 99% da imagem**. Este teste deve fornecer uma idéia do desempenho dos algoritmos relativo ao número de pixels de interesse. A TDE por força bruta irá demorar mais para 50% de pixels, como analisado na Seção 2.1. Portanto, a primeira vista, pode-se esperar um comportamento geral similar para os outros algoritmos.

5. **Uma linha de pixels pretos de** $0°$ **a** $90°$ **passando pelo centro da imagem**. O pior caso dos algoritmos de propagação ocorre para inclinações diferentes de horizontal, vertical ou diagonal, como explicado por Ragnemalm [15]. Este teste foi utilizado por Cuisenaire para algumas comparações.

6. **Quadrados aleatórios**. Estas imagens são geradas escolhendo-se aleatoriamente os centros e tamanhos de quadrados pretos, rotacionados por $\theta \in [0, 90°]$. Os quadrados são preenchidos e desenhados na imagem até que o número de pixels pretos ultrapasse uma porcentagem p. Uma imagem desse tipo será denotada pelo par (p, θ). Este teste foi sugerido por Eggers, pois utiliza uma imagem sintética que possui alguma semelhança com imagens reais orientadas. Esta imagem é mais realística que as imagens de uma única linha para testar o desempenho com respeito à orientação.

7. **Imagens binarizadas de objetos reais**. Pretende-se utilizar a tradicional imagem da Lenna[1] com bordas detectadas e limiarizadas. A Lenna foi escolhida pois tem sido utilizada como um *benchmark* universal e relativamente imparcial para algoritmos em análise de imagens. Futuramente, também pretende-se executar os algoritmos em imagens binárias de letras, folhas de plantas e neurônios. As imagens de letras foram são interessantes pois OCR é uma aplicação bastante popular de análise de imagens. Já as folhas de plantas fornecem uma grande variedade de formas: finas, grossas, arredondadas, crespas, etc. Por fim, neurônios são bastante finos e possuem perímetro bastante grande. Dessa forma, pretendemos testar as TDEs para uma variedade ampla e significativa de formas.

Além de testes com as imagens acima, também pretende-se, futuramente, extrair medidas das imagens para caracterizar o desempenho dos algoritmos de acordo com o a forma do objeto de entrada, como descrito nos objetivos (Seção 1.2).

3.2 Procedimento de Teste

Os testes foram realizados conforme descrito no Algoritmo 9 a seguir. Os resultados do teste são colocados em arquivos em texto legível e bem estruturado, de forma a permitir uma extração dos

[1] A história desta imagem pode ser encontrada na Internet, em www.lenna.org

dados com utilitários baseados em expressões regulares. Gráficos de desempenho foram gerados em Scilab, usando-se tabelas construídas a partir desses resultados brutos.

Algoritmo 9 Teste principal de desempenho e exatidão

Para cada imagem teste e para cada algoritmo:

1. Executar o algoritmo, medindo o tempo. Se a imagem for suficientemente pequena, executar esta etapa várias vezes até obter uma boa precisão na medição.
2. Se o tempo de execução for muito alto (maior que $30s$), armazenar a o mapa resultante em um arquivo para futura análise de exatidão.
3. Calcular uma TDE de referência para avaliar a exatidão:
   - Se a imagem for suficientemente pequena, executar o força-bruta. Salvar o resultado em um arquivo para futuros testes.
   - Se a imagem for grande, a referência será a TDE calculada por algoritmo rápido que tem se mostrado exato em todos os testes. No entanto, se a imagem for demasiadamente grande, verificar se já não existe um arquivo com uma TDE de referência pré-calculada. Se não existir, calcular e armazenar a TDE de referência num arquivo para testes futuros.
4. Comparar o mapa do algoritmo sendo testado com o mapa de referência. Se houver erro, apresentar as seguintes medidas:
   - Número de pixels errados e $\%err = \frac{\text{número de pixels errados}}{\text{número de pixels brancos}}$
   - Máximo erro e $\%err_{max} = \frac{\text{máximo erro}}{\text{máxima distância}}$

No Passo 4 do Algoritmo 9, a TD de referência utilizada foi o algoritmo de Maurer, que se mostrou exato em testes preliminares.

Neste trabalho, não será avaliado empiricamente o uso de memória pelos algoritmos. Entretanto, esta tarefa não é muito difícil de ser realizada em sistemas Unix. Uma maneira de medir o uso de memória em tempo de execução é através de chamadas de sistema ou, mais facilmente, através de utilitários que fornecem o uso de memória de determinado processo. Outra forma é fazer uso do sistema de arquivos `'/proc'` do Linux. Um modo mais complexo porém preciso seria utilizar bibliotecas ou utilitários de rastreamento de memória que substituem as funções `malloc`, da linguagem C, por outras que rastreiam a quantidade de memória utilizada, por exemplo as bibliotecas e utilitários `ccmalloc` [83] e `valgrind` [84].

Na medida em que forem sendo obtidos os resultados empíricos, deverá ser realizada uma ponte com a teoria. Será estudado o motivo teórico do comportamento verificado nos resultados, com possíveis demostrações desses comportamentos; e vice-versa: foram comprovados empiricamente resultados teóricos e conjecturas. Alguns exemplos de perguntas específicas a serem respondidas por um tal estudo são:

- O novo método de Cuisenaire [6] é realmente exato? Ele possui comportamento linear mesmo para imagens não testadas no artigo original do autor? Se os experimentos mos-

trarem evidência de respostas afirmativas, haverá uma boa razão para tentativas futuras de demonstrar tais fatos em teoria. Se os experimentos apontarem o contrário, o erro da implementação ou teoria foram procurados.

- O algoritmo de Saito [7] é $O(n^3)$? Este caso é freqüente?
- O algoritmo recente de Maurer [8] é realmente rápido e exato?

3.3 Outras Propriedades Analisadas

Além do desempenho e exatidão, outro critério importante para comparar algoritmos de TDEs é a facilidade de implementação. Um método $O(n^3)$ no pior caso pode ser preferível se for razoavelmente rápido no caso médio e muito mais fácil de implementar que um método ótimo. Para ajudar a avaliar a facilidade de implementação, a princípio será relatado o número de linhas da implementação de cada TDE por uma mesma pessoa, assim como as estruturas de dados utilizadas. Porém, a verdadeira análise deste critério será pessoal, com argumentos razoavelmente bem fundamentados na experiência.

Capítulo 4

Resultados Empíricos

Os testes foram realizados em um PC com processador Intel Pentium 4 1.7GHz, 1GB de memória Rambus RDRAM, com sistema operacional Slackware Linux 9.1, Kernel v2.4.26. Os algoritmos foram compilados com GCC v3.2.3 sem *flags* de otimização, e serão disponibilizados na biblioteca Animal [49] assim que os resultados dessa dissertação forem publicados em periódico. Para a automação da bateria de testes, foram utilizados *scripts Bash*, um intepretador comum em sistemas Unix, para coordenar a execução de programas em C e gerar os arquivos de resultados. A exibição dos dados empíricos também foi programada na linguagem do ambiente Scilab [85]. Além disso, a temporização do código foi implementada utilizando-se a função ANSI-C `clock`, que mede o tempo de CPU, permitindo que os testes sejam confiáveis mesmo quando executados em *timesharing*.

Neste texto, serão apresentados os gráficos mais relevantes, com foco nos tamanhos grandes de imagens (n da ordem de 10^3). Os resultados completos estão disponíveis na web [86], e o *framework* de testes também será liberado como software livre sob licença GPL (GNU *Public License*). Porém, esse material não será tornado público até que seja publicado. A senha de acesso ao website com os resultados e implementação será concedida à banca examinadora mediante um pedido ao autor desta dissertação.

4.1 Imagem com um ponto no canto

Como mostra o gráfico da Figura 4.1, o melhor desempenho ocorreu para Eggers e Maurer, estando Lotufo-Zampirolli e Saito bastante próximos e com a mesma ordem de complexidade. A grande surpresa deste teste foi o desempenho de Cuisenaire: o mais lento para este caso, quase 7 vezes mais lento que Maurer[1] e com complexidade aparentemente superior à dos outros algoritmos. Outro fato interessante é Eggers ter desempenhado bem neste teste, sendo um algoritmo de propagação $O(n^3)$, ao passo que Cuisenaire também é um algoritmo de propagação supostamente $O(n^2)$.

[1] Neste texto, esse tipo de comparação de velocidades foi realizado para a maior imagem do teste em questão, caso nada diferente seja afirmado.

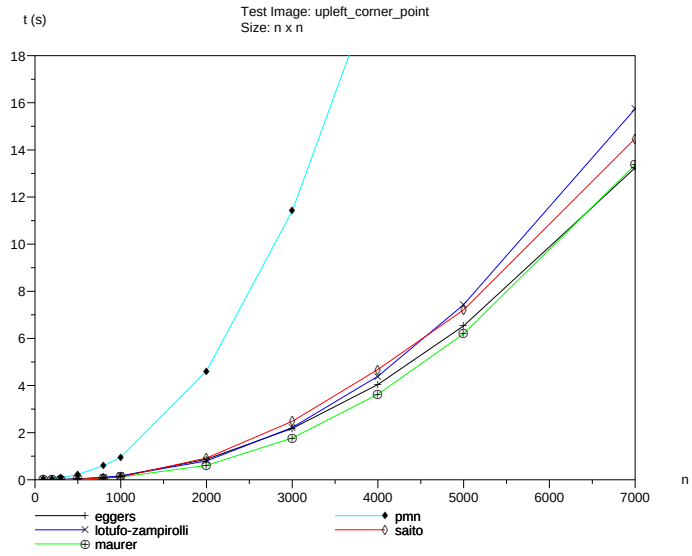


Figura 4.1: Desempenho para a imagem contendo um único ponto preto no canto superior esquerdo.

4.2 Círculo

Maurer, Cuisenaire e Saito foram bastante próximos neste teste, como mostra o gráfico da Figura 4.2. Bem mais lentos, Eggers e Lotufo-Zampirolli desempenharam em torno de 10 e 6 vezes mais lentos que Maurer, respectivamente.

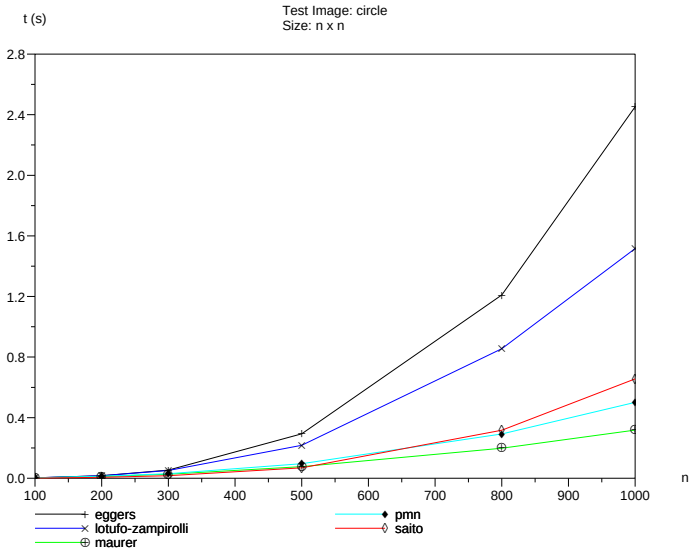


Figura 4.2: Desempenho para a imagem contendo um círculo branco inscrito.

4.3 Bordas de Lenna

O gráfico da Figura 4.3 indica que todos os métodos apresentam a mesma complexidade para este conteúdo. Novamente, Maurer apresenta o melhor desempenho, seguido por Saito. Em seguida, vieram Cuisenaire e Eggers bastante próximos. O método mais lento neste teste foi o de Eggers, aproximadamente três vezes mais demorado que Maurer.

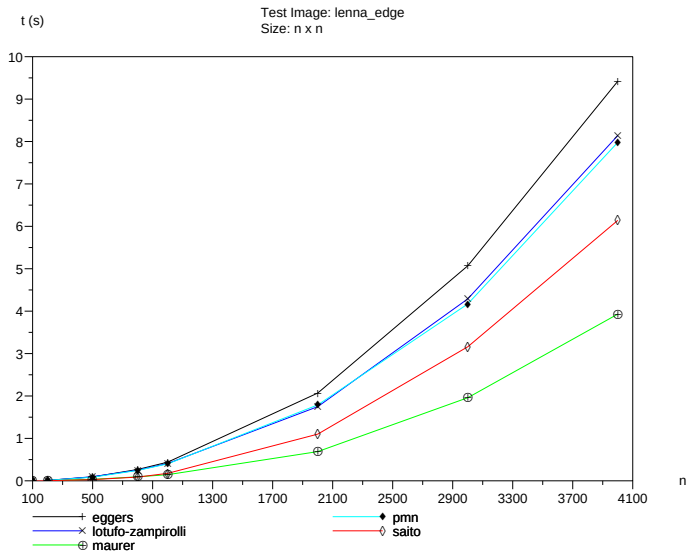


Figura 4.3: Desempenho para a imagem contendo bordas da Lenna (distâncias calculadas fora).

4.4 Imagem meia-preenchida

Como mostra a Figura 4.4, este é o melhor desempenho relativo de Eggers e o pior de Maurer, para todos os conteúdos de imagens testados. Eggers é seguido por PMN e Saito, que estão próximos, por suas vezes seguidos de Lotufo-Zampirolli e, finalmente, Maurer. Os algoritmos de propagação desempenharam melhor que os algoritmos de varredura raster para esta imagem. Apesar de Maurer ter sido o melhor método para diversos tipos de imagens, neste caso ele foi aproximadamente 5 vezes mais lento que Eggers.

4.5 Pixels aleatórios

O número de pixels de interesse se mostrou um fator de grande impacto na performance dos métodos. Como mostra a Figura 4.5, a velocidade de quase todos algoritmos é proporcional à quantidade de pixels brancos. Eggers constitui a única excessão – possui um pico de tempo em torno de 60%, e a melhor performance continua sendo perto de um único pixel branco. Esse pico em torno de 60% é similar ao comportamento do algoritmo força-bruta, que tem seu pico em 50%.

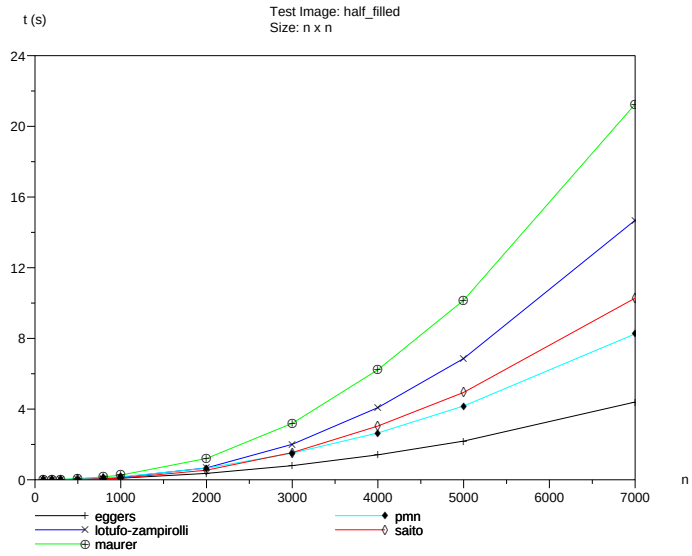


Figura 4.4: Desempenho para a imagem com a metade superior composta de pixels pretos.

A 100×100, os algoritmos mais rápidos foram Saito, a partir de 10%, e PMN e Eggers, antes de 10%. Já a 4000×4000, Saito se aproxima de PMN, ambos sendo os mais rápidos entre 15% e 75%. Mas Eggers passa a ser o mais rápido fora deste intervalo.

O algoritmo de Maurer foi o mais lento até 90%, posição tomada por Cuisenaire a partir de 90% para todos os tamanhos menores que 4000×4000. Entretanto, Maurer foi o método mais estável, seguido por Lotufo-Zampirolli. Saito vai ficando mais instável quanto maior a imagem. Os menos estáveis foram os métodos de Eggers e PMN, sendo um pouco mais estáveis quanto maior a imagem. Saito fica mais instável com o aumento da imagem, aproximando-se de PMN.

4.6 Linha giratória

Todos os métodos se mostraram dependentes da orientação. Como mostra o gráfico da Figura 4.6, os mais estáveis foram Maurer e PMN, sendo este último até três vezes mais lento que o primeiro. Já os algoritmos menos estáveis foram os de Eggers, Lotufo-Zampirolli e Saito. Na medida em que o tamanho da imagem cresce, tem-se a impressão de PMN e Maurer parecerem mais estáveis pois vão se tornando muito mais rápidos que os outros métodos. Porém o comportamento permanece o mesmo, como mostra a ampliação do gráfico da Figura 4.6(d) na Figura 4.7.

A diferença de desempenho entre cada um dos métodos é bastante acentuada para inclinações diferentes de 0° e 90°, principalmente para os ângulos entre 60° e 70°. O desempenho são simétricos em relação a 90°. Claramente, Saito tem seu pior desempenho para 60°. Eggers apresenta picos de lentidão em torno de 20° e 75°, e mínimos em torno de 0° e 45°. Já o PMN é mais lento para 45° (diferentemente de Eggers) e mais rápido em 0°. Lotufo-Zampirolli apresentou picos em torno de 55° e máximo em 0°. Curiosamente, Maurer apresenta um mínimo em 90° e máximo constante entre 0° e 40°. Essa assimetria de Maurer foi introduzida devido à

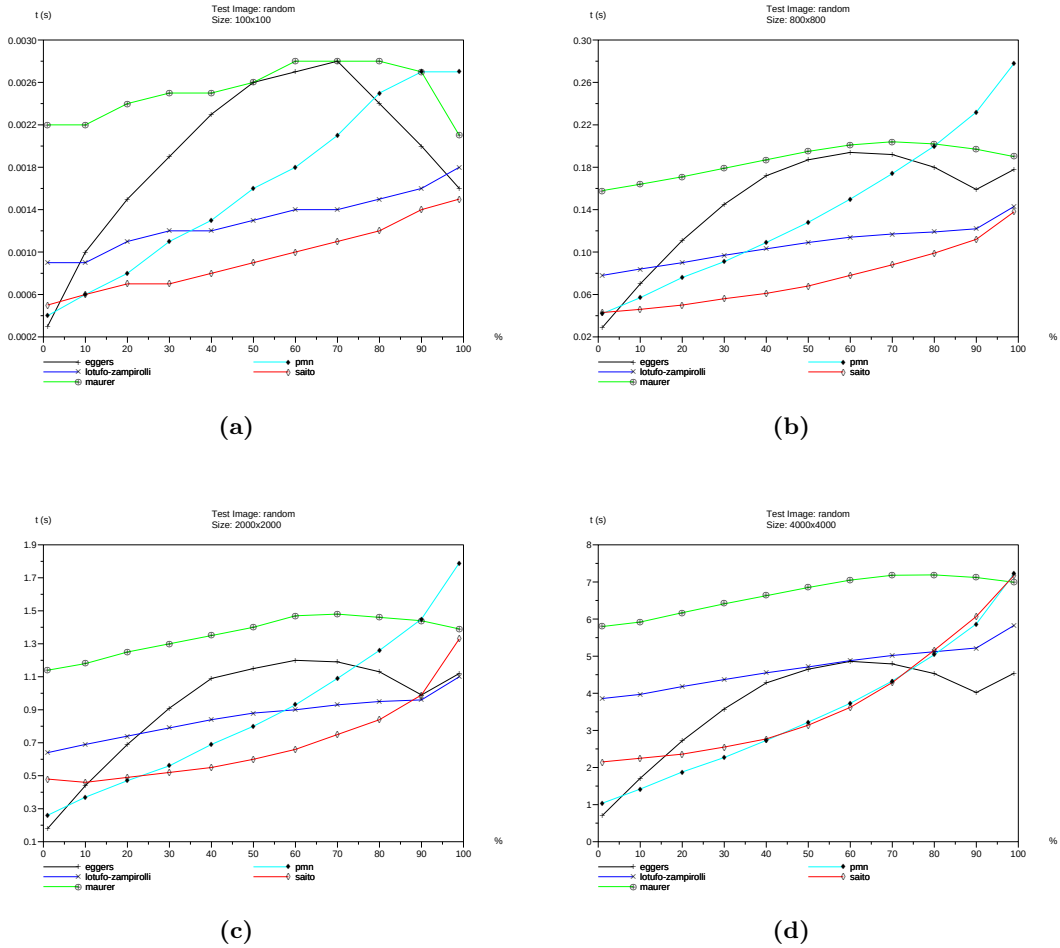


Figura 4.5: Desempenho dos métodos para imagens com porcentagem variada de pixels brancos aleatóriamente espalhados. Os tamanhos aqui mostrados são 100×100 (a), 1000×1000 (b), 3000×3000 (c) e 4000×4000 (d).

escolha de processamento de colunas seguido de linha, em vez do contrário. Os outros algoritmos de varredura raster possuem essa assimetria de linha/coluna, porém não apresentaram diferença significativa entre o desempenho para as orientações vertical e horizontal.

Para imagens menores que 500×500, Saito teve o melhor desempenho médio, sendo. Para o restante, Maurer foi o mais rápido. A evolução das curvas relativamente ao tamanho das imagens evidenciam que Lotufo-Zampirolli, Eggers e Saito não são lineares fora de seus tempos mínimos.

4.7 Quadrados aleatórios

Em todos os gráficos deste teste, mostrados nas Figuras 4.8 e 4.9, observa-se uma forte dependência dos métodos em relação à porcentagem de pixels. A única exceção é o método de Maurer, que se mostrou muito mais estável que todos os outros métodos, para todos os ângulos.

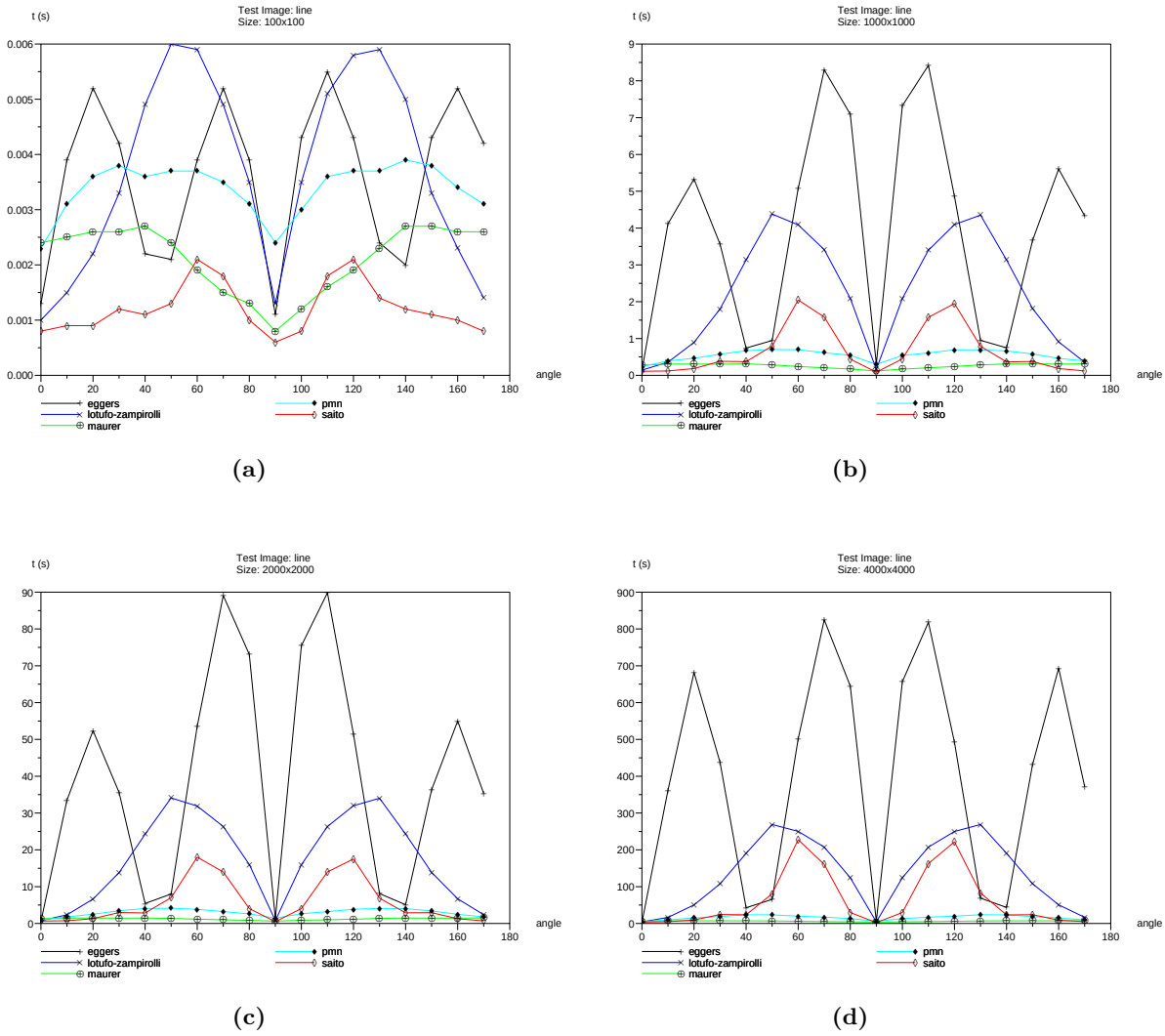


Figura 4.6: Desempenho dos métodos para imagens com uma linha de inclinação variada. Os tamanhos aqui mostrados são 100×100 (a), 1000×1000 (b), 3000×3000 (c) e 4000×4000 (d).

No entanto, Maurer raramente foi o mais rápido. Invariavelmente, Maurer foi o método mais lento para os ângulos 0° (e 90°), para porcentagens maiores que 15%. Para as porcentagens altas de pixels pretos, o método foi o mais lento sob qualquer orientação. Isto pode ser verificado, por exemplo, a partir dos gráficos para porcentagem fixa igual a 95%. Para qualquer ângulo fixo, observa-se que todos os outros métodos são mais rápidos para uma maior quantidade de pixels pretos, pois as as curvas de ângulo fixo são decrescentes.

Os gráficos foram todos simétricos em relação a 45°. Ademais, Para porcentagem alta de pixels de interesse, 95%, observa-se uma menor dependência dos métodos em relação ao ângulo. De fato, ao variar-se a orientação da imagem com poucos pixels brancos, pouco se altera a distribuição de distâncias a serem calculadas.

Assim como no teste das retas inclinadas, Lotufo-Zampirolli e Eggers foram os mais depen-

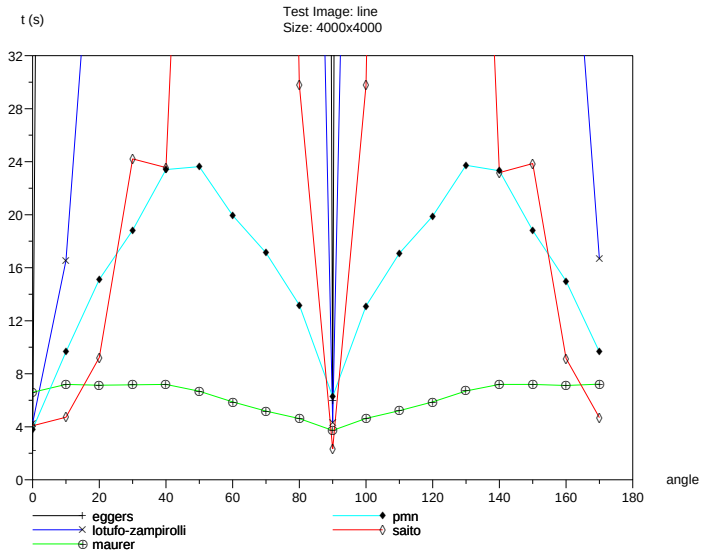


Figura 4.7: Ampliação do gráfico da Figura 4.6(d).

dentes da orientação. Cuisenaire se mostrou bastante dependente em relação à porcentagem, principalmente para ângulos perto de 0°.

Para imagens grandes, Lotufo-Zampirolli apresentou a pior média a 45° e Eggers a 15°, exceto a porcentagens altas de pixels pretos.

4.8 Exatidão

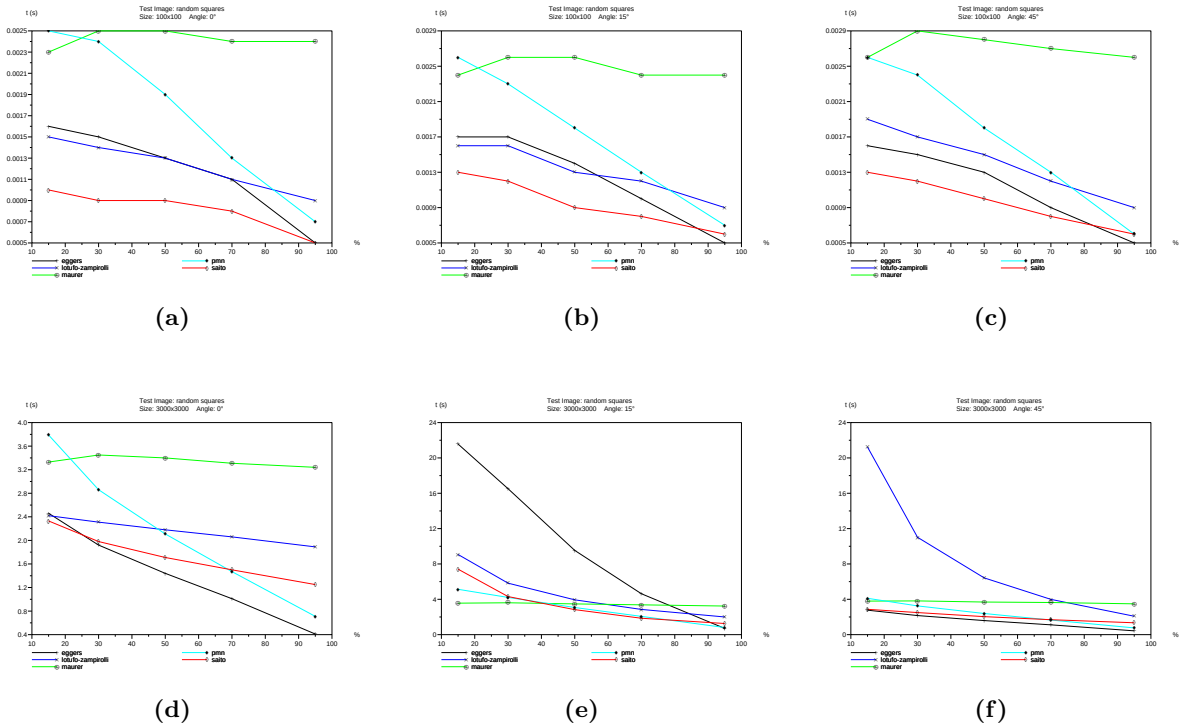
Todos os métodos se mostraram exatos segundo o teste descrito pelo algoritmo 9 da seção 3, exceto pelo PMN de Cuisenaire implementado pelo autor desta dissertação.

De todos os testes, as imagens para as quais a implementação do PMN apresentou erro foram: quadrados aleatórios, círculo 300×300, Lenna a partir de 500×500 e a reta para todos os ângulos exceto 0° e 90°. Ainda não foram econtrados erros na implementação do PMN, mas não se sabe se ela está correta. Futuramente, pretende-se estudar se estes erros são falhas do método ou da implementação. Acredita-se que a liberação do código fonte da implementação do PMN utilizada nos testes contribuirá para seu aperfeiçoamento e melhor validação.

De todos os testes, a implementação do PMN utilizada apresentou no máximo 0.04% pixels errados (4 em 9900), para a imagem 100×100 da reta com aproximadamente 60°. O máximo erro foi de 0.15% da distância máxima para a imagem de quadrado aleatório $(70\%, 30°)$.

4.9 Discussão dos Resultados Empíricos

Os resultados se mostraram bastante variados, sendo que nenhum método foi mais rápido em todos os testes. Um exemplo da diversidade dos resultados é que, apesar das bordas de Lenna parecem ser um caso de imagem aleatória com baixa porcentagem de pixels pretos, comparando-



(a) **(b)** **(c)**

(d) **(e)** **(f)**

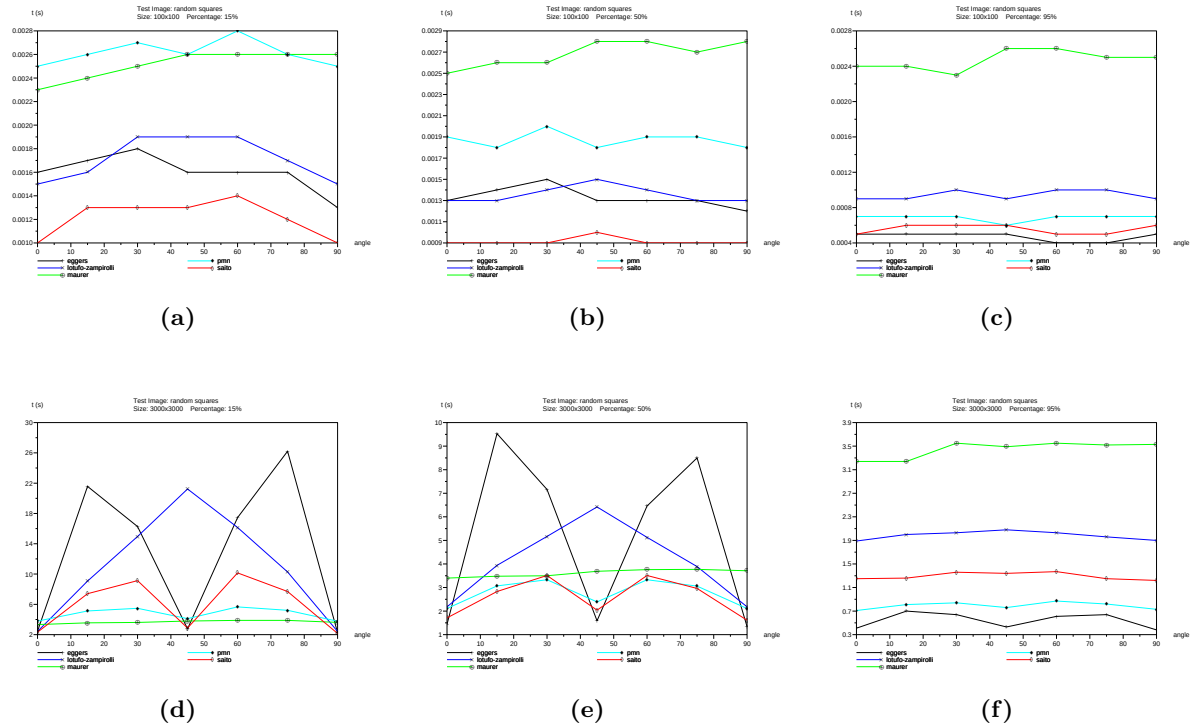
Figura 4.8: Desempenho dos métodos para imagens de quadrados aleatórios, variando-se a porcentagem (abcissas) e fixando-se a orientação nos valores 0° em (a) e (d), 15° em (b) e (e), e 45° em (c) e (f), para imagens 100×100 (fileira superior) e 3000×3000 (fileira inferior).

se os gráficos para os testes de pixel aleatórios e quadrados, nas Figuras 4.5 e 4.8, obtêm-se comportamentos bastante diferentes. Portanto, a velocidade dos métodos de TDE testados de fato não depende apenas da quantidade de pixels de interesse, mas também da disposição ou geometria desses pixels.

Nota-se também a diferença entre os desempenhos de uma imagem aleatória com 50% de preenchimento e das imagens aleatórias com 50% de pixels ou de 50% de quadrados. Na imagem meia-preenchida, as distâncias a serem calculadas são bem maiores que as distâncias em uma imagem com os pixels de interesse uniformemente distribuídos. Ademais, curiosamente, o pico de Eggers a 60% da Figura 4.5 não ocorre para a imagem de quadrados aleatórios para todo ângulo fixo (Figura 4.8).

A velocidade dos métodos é proporcional ao número de pixels de interesse, ou seja, inversamente proporcional à quantidade de pixels brancos (onde as distâncias são calculadas). Isto pode ser observado, por exemplo, no gráfico de pixels aleatórios da Figura 4.5, onde todos as curvas possuem uma tendência ascendente na medida em que aumenta-se a porcentagem de pixels brancos. Este comportamento também pode ser verificado na Figura 4.8, onde as curvas são decrescentes em relação ao número de pixels pretos.

Os fatores constantes na função temporal de um algoritmo podem ser consideravelmente reduzidos apenas otimizando a implementação. Portanto, os gráficos devem ser analisados princ-



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figura 4.9: Desempenho dos métodos para imagens de quadrados aleatórios, variando-se o ângulo (abcissas) e fixando-se a porcentagem de pixels pretos nos valores 15% em (a) e (d), 50% em (b) e (e), e 95% em (c) e (f), para imagens 100×100 (fileira superior) e 3000×3000 (fileira inferior).

palmente pela forma das curvas, e menos pelo valor absoluto de desempenho. Ademais, atenção especial deve ser dada aos valores de desempenho para imagens grandes, casos em que os fatores constantes têm menos influencia entre algoritmos de complexidades diferentes.

Todos os testes de ângulos se mostraram simétricos em relação a 90°, sendo desnecessário testar para ângulos fora do intervalo $0° - 90°$. No entanto, nenhum método apresentou simetria em relação a 45° no intervalo $0° - 90°$, exceto por Eggers, que apresentou essa simetria para imagens de reta e para imagens grandes de quadrados aleatórios.

A seguir são discutidos alguns resultados para cada algoritmo.

4.9.1 Desempenho do PMN de Cuisenaire

O pior caso de Cuisenaire para todos os testes até o momento ocorre para imagens com muitos pixels brancos. Tal fato é visível no gráfico para a imagem de um pixel preto no canto (Figura 4.1), bem como nos gráficos de porcentagem variada (Figuras 4.5 e 4.8): quanto mais pixels brancos, pior o desempenho de Cuisenaire. Cuisenaire parece ser o método mais dependente do número de pixels de interesse.

O método se mostrou relativamente estável à variação de ângulo, ficando atrás apenas de Maurer. Como evidenciado nos gráficos das Figuras 4.6 e 4.9, PMN apresenta melhor desempe-

nho para 0° ou 90°, piorando na medida em que a inclinação se afasta desses valores. Ademais, é o único método além de Maurer que parece ter complexidade linear relativo ao tamanho da imagem da reta e dos quadrados com inclinação variada.

Conforme relatado na seção 4.8, os testes não confirmaram a exatidão do método para diversas imagens. Está ainda por ser verificado se o erro está na implementação, que é relativamente elaborada, ou na própria teoria do método. Se a implementação estiver correta, restará estudar os casos de erro e mostrar por que não foram corrigidos pelas propagações múltiplas.

4.9.2 Desempenho de Maurer

O algoritmo de Maurer mostrou ser o mais estável relativo ao conteúdo da imagem, como verificado nas Figuras 4.5, 4.6, 4.8 e 4.9. Entretanto, Maurer apresentou alguma dependência ao conteúdo, principalmente relativo ao teste da reta inclinada. Como mostra a Figura 4.6, seu melhor desempenho ocorreu para 90°, que, curiosamente, foi mais rápido que o caso de 0°. Porém essa dependência à inclinação não foi verificada no teste dos quadrados aleatórios, como mostra o gráfico na Figura 4.9.

O algoritmo de Maurer também mostrou ser o mais rápido para grande parte das imagens: bordas de Lenna (Figura 4.3); círculo inscrito (Figura 4.2); ponto no canto (Figura 4.1); a maior parte das linhas inclinadas (Figura 4.6); e quadrados com baixa porcentagens (Figuras 4.8 e 4.9).

Porém, não se pode dizer que Maurer é o melhor algoritmo de TDE para todos os casos, pois várias vezes se mostrou mais lento que todos os outros métodos. Isto ocorreu para as imagens aleatórias com porcentagens altas de pixels de interesse, como mostrado nas Figuras 4.5 e 4.8, bem como para a imagem meia-preenchida. No entanto, nesses casos Maurer não ficou muito atrás dos outros métodos. Por exemplo, nunca ficou mais do que 6 vezes mais lento que Saito, caso que ocorreu para o quadrado $(95\%, 60°)$, como mostra a Figura 4.9(f).

4.9.3 Desempenho de Saito

O método de Saito desempenhou bem na média; não foi o mais lento em nenhum caso e é fácil de implementar. Porém, para o seu pior caso – a linha inclinada a 60° – chega a ser 40 vezes mais lento que Maurer. Já Maurer não chegou a ficar mais que 6 vezes mais lento que Saito, caso que ocorreu para o quadrado $(95\%, 60°)$.

O teste da reta inclinada (Figura 4.6) mostra que a complexidade de Saito realmente é pior que $O(n^2)$. Isso fica evidente para imagens de tamanhos grandes, onde o método de Saito se destaca dos métodos de Maurer e Cuisenaire. No entanto, para todos os demais testes o método parece ser $O(n^2)$, já que o crescimento da sua curva relativo ao tamanho da imagem se aproxima dos demais algoritmos $O(n^2)$.

Em relação à orientação do conjunto de interesse, o método se mostrou bastante dependente, se comparado aos métodos de Maurer e Cuisenaire. Os piores tempos foram verificados para orientações em torno de 60° tanto para o teste da reta (Figura 4.6) como dos quadrados (Figura 4.9).

Em relação à quantidade de pixels de interesse, o algoritmo de Saito se mostrou bastante dependente. No teste de pixels aleatórios, quanto maior a imagem, mais aguda a curva de desempenho relativa à porcentagem de pixels brancos (Figura 4.5), se aproximando muito da curva de Cuisenaire. O desempenho se foi melhor para muitos pixels de interesse. Para o teste dos quadrados, verificou-se um comportamento semelhante, como mostra a Figura 4.8.

4.9.4 Desempenho de Lotufo-Zampirolli

O método teve um desempenho mediano em relação aos outros algoritmos testados. Foi razoavelmente estável quanto à porcentagem de pixels de interesse, como evidenciado na Figura 4.5. A excessão constituiu o teste dos quadrados aleatórios em torno de 45°, como mostra a figura Figura 4.8(f).

O algoritmo de Lotufo-Zampirolli claramente não se mostrou linear no teste da reta inclinada, para os ângulos não-retos. Ademais, foi o segundo método mais dependente da inclinação da reta e dos quadrados aleatórios. Seu pior desempenho ocorreu em torno de 45°, tanto no teste dos quadrados como no das retas. Neste caso, o desempenho foi também o pior em relação aos outros métodos, como mostra a figura 4.8(f), exceto a partir de 80% de pixels pretos. No entanto, a tendência observada nos gráficos similares para todos os tamanhos testados leva-nos crer que, para imagens maiores que 3000×3000 a 45°, o método foi o pior para praticamente todas as porcentagens de pixels pretos. Constatou-se, também, que a implementação utilizada não mostrou melhor desempenho relativo para nenhuma imagem testada.

4.9.5 Desempenho de Eggers

Eggers foi o algoritmo mais rápido para a imagem meia-preenchida, fato confirmado pela imagem similar formada de quadrados aleatórios com 50% de pixels de interesse a 0°. Para a imagem com um ponto de interesse no canto o algoritmo também conquistou o primeiro lugar, empatando com Maurer. Entretanto, Eggers foi o método mais lento para as bordas de Lenna, para o Círculo inscrito e para diversos imagens dos outros testes, como analisado a seguir.

O método de Eggers foi o mais dependente do conteúdo, tanto da quantidade de pixels de interesse como da orientação. No teste dos pixels aleatórios, o algoritmo apresentou picos de tempo em torno de 50% de pixels brancos, melhorando drasticamente para baixa porcentagens e para altas porcentagens em torno de 90%. No entanto, para acima de 90% de pixels brancos o método apresentou outra queda na eficiência, como mostra a Figura 4.5. O mesmo tipo de comportamento não foi verificado no teste dos quadrados: para todo ângulo fixo a curva foi monótona, sendo mais baixa para baixas porcentagens de pixels brancos.

Em relação ao ângulo, o método também foi o menos estável. Para as imagens da linha inclinada, o método apresentou máxima velocidade em torno de 0° e 45°, sendo bastante lento fora desse intervalo, principalmente para os ângulos entre 60° e 80°. Esse fato também foi verificado no teste dos quadrados aleatórios com tais inclinações e com porcentagens de pixels de interesse inferior a 80%, como mostra a Figura 4.8(e).

Capítulo 5

Conclusões e Perspectivas

Neste trabalho, os principais algoritmos avançados de Transformada de Distância Euclidiana foram descritos, implementados e comparados de forma inédita. Os testes empíricos mostraram a grande variação no desempenho dos algoritmos de TDE em relação ao conteúdo da imagem. Nenhum método se mostrou definitivamente mais rápido em todos os casos testados.

O método de Maurer mostrou ser a melhor opção dentre os algoritmos estudados. Apresentou um excelente desempenho relativo na maioria dos testes, com certeza o desempenho mais independente do conteúdo da imagem. A descrição fornecida nesta monografia contribui para seu melhor entendimento, e a implementação de código livre realizada durante este mestrado também contribuirá para uma maior adoção deste algoritmo de TDE.

Saito pode ser considerado uma boa alternativa ao algoritmo de Maurer. Apesar de ser $O(n^3)$, mostrou um desempenho relativo excelente para todos os casos diferentes do pior caso. Além disso, o algoritmo é de fácil implementação, sem dúvida o menos complexo de todos os algoritmos considerados nesta monografia. Sua extensão ao 3D é imediata.

A grande surpresa nestes estudos foi o desempenho de Cuisenaire, que foi abaixo do esperado. Os testes apontaram erros nos mapas produzidos pela implementação testada, mas ainda não se sabe se o erro é teórico. Uma vantagem importante de Cuisenaire é a possibilidade imediata de gerar a TDE até uma distância máxima apenas.

Os algoritomos de Eggers e Lotufo-Zampirolli mostraram um desempenho inferior ao de Saito. Ambos são bastante dependentes do conteúdo da imagem, e os testes confirmaram suas complexidades $O(n^3)$.

No geral, os algoritmos de varredura independente se mostraram mais rápidos e fáceis de implementar que os demais algoritmos testados.

5.1 Contribuições

As principais contribuições deste trabalho são:

- **Levantamento bibliográfico** atualizado, extenso, comentado e categorizado. Uma base eletrônica de referências categorizadas e revisadas foi montada pelo aluno, e será divulgada

quando o artigo relacionado a esta dissertação for aceito para publicação.

- **Descrição dos algoritmos** de maneira uniforme e mais conceitual que aquela realizada pelos autores originais. Espera-se, também, que a explicação dos algoritmos dada neste trabalho tenha elucidado passagens obscuras dos artigos originais. As descrições dos algoritmos de Maurer, Cuisenaire e IFT são contribuições inéditas e especialmente relevantes.

- **Validação dos algoritmos de TDEs**. Ainda não havia sido realizado um estudo independente de validação dos algoritmos recentes de TDE.

- **Implementação confiável de TDEs**. O código em linguagem C, a ser disponibilizado em software livre, aumentará ainda mais a aplicabilidade e confiabilidade dos métodos de TDEs.

- **Implementação acessível de TDEs**. Foi desenvolvida uma interface amigável para o Scilab de métodos recentes e confiáveis [50], a qual será divulgada em breve.

- **Caracterização dos principais métodos TDEs**, realizado por um estudo empírico e teórico de suas propriedades.

- **Fornecimento de algoritmos mais confiáveis e comprovadamente eficientes** para resolver não só a TDE, mas também e outros problemas baseados em propagação *Eikonal*.

- **Potencial para futuras contribuições teóricas**, tais como demonstrações de propriedades e conjecturas nas teorias dos algoritmos de TDE e o desenvolvimento de novos métodos. Os experimentos forneceram evidência para comportamentos e conjecturas a serem provados futuramente.

- **Metodologia útil e extensível para comparar algoritmos de TD**. A experiência adquirida neste trabalho ajudará, sem dúvida, na avaliação de novos algoritmos e dos que não foram abordados. Algoritmos relacionados – que utilizem outras métricas e outros domínios – também podem ser avaliados com uma metodologia semelhante.

- **A validação de novos algoritmos foi facilitada.** Espera-se que novos algoritmos de TDE sejam facilmente validados, dada a metodoligia e disponibilidade da implementação de TDEs proporcionados por este trabalho.

- **A listagem inédita de dados sobre erros para vizinhanças maiores que** 31×31 **na Tabela 2.1**. Isso permite aplicar o método a vizinhanças maiores que originalmente possível usando as tabelas originalmente publicadas. Também foi constatado que a linha 3 está errada tanto na tese como no artigo de Cuisenaire [43, 6]. A linha 16 foi publicada com erro apenas na tese de Cuisenaire, mas está correta em seu artigo. Como a Tabela 2.1 é utilizada diretamente pelo algoritmo de Cuisenaire, essas constatações influenciam a exatidão do método de Cuisenaire.

Espera-se que este trabalho seja amplamente útil para a comunidade de geometria e visão computacionais, já que a TDE é a base de diversos outros operadores, técnicas e aplicações.

5.2 Desenvolvimentos Futuros

Diversas atividades decorrem naturalmente do presente trabalho, dentre elas:

- **Extensão para outros problemas.** Por exemplo, os melhores métodos devem ser estendidos para a propagação de rótulos, permitindo o cálculo de diagramas de Voronoi discretos, eixos mediais multi-escala, dentre outros.

- **Extensão para outros domínios.** TDs podem ser estendidas para solucionar diversos problemas importantes, como planejamento de trajetórias mínimas em domínios não-convexos ou em superfícies.

- **Extensão para outras métricas.** As idéias expostas neste trabalho podem eventualmente serem aplicadas para outras métricas, inclusive métricas geodésicas que levam em conta níveis de cinza ou cor. Desta forma, pode-se pensar utilizar as idéias das TDEs rápidas para problemas como segmentação de imagens, da mesma forma como já é feito com a IFT.

- **Avaliação empírica dos principais métodos em 3D**. O processamento de imagens volumétricas tem sido cada vez mais comum. Por envolver uma enorme quantidade de dados, a eficiência das TDEs em 3D é ainda mais importante que para imagens 2D.

- **Inclusão do *fastmarching*** [87] no estudo comparativo realizado neste trabalho. Trata-se de uma técnica bastante utilizada, modelada por equações diferenciais parciais.

- **Estudo teórico aprofundado** dos melhores algoritmos. É importante demonstrar propriedades ainda não provadas, ou ao menos estudar detalhadamente as demonstrações de tais propriedades, validando-as. Deve-se realizar uma análise algoritmica detalhada.

- **Descobrir as causas do comportamento empírico.** Por exemplo, deve-se descobrir por que o método de Cuisenaire apresentou erros.

- **Incorporar mais imagens teste.** A única imagem real testada foi a das bordas de Lenna. Deve-se repetir o teste para bordas de outras imagens reais. Para tanto, será importante determinar tipos de imagens a serem testadas e verificar o comportamento para representantes de cada tipo.

- **Incorporar outros métodos no teste.** Por exemplo, poderia ser incluído na comparação os métodos não-euclidianos Chamfer de Borgefors [42, 24] e o método inexato de Danielsson [19]. A nova comparação poderia envolver esses dois algoritmos e apenas os métodos de Maurer e Saito.

- **Mostrar formalmente que o método de Shih [9] está errado.** Caso contrário, compará-lo com os métodos de Maurer e Saito.

Conquistas Alcançadas no Mestrado

Foi escrito um artigo [88] sobre a *toolbox* SIP desenvolvida pelo autor [50]. O artigo foi aceito para publicação na Linux Journal. Um artigo em forma de Survey relativo a este trabalho de mestrado está sendo finalizado e será submetido para revista internacional. O autor deste trabalho conseguiu uma bolsa do CNPq para realizar doutorado na Brown University, EUA. Como a bolsa tem início em Setembro de 2004, este projeto de mestrado terminou mais cedo que o previsto. A duração do mestrado foi de 1 ano: 6 meses matriculado, juntamente com 6 meses como aluno especial. Nos EUA, o autor deverá continuar sua pesquisa em análise de formas, dando continuidade ao presente trabalho no âmbito mais geral de evolução de fronteiras. As técnicas aqui estudadas serão aplicadas e estendidas, juntamente com técnicas de equações diferenciais parciais, para resolver problemas em análise de imagens. Tais problemas vão desde a TD, esqueletos e dimensão fractal, até segmentação, busca de trajetórias mais curtas e filtragem de ruídos. Pretende-se manter uma cooperação bastante próxima com os pesquisadores brasileiros.

Bibliografia

[1] A. Rosenfeld and J. Pfaltz, "Sequential operations in digital picture processing," *Journal of the ACM*, vol. 13, no. 4, 1966.

[2] F. John, *Partial Differential Equations*. Universitext, Springer-Verlag, 4th ed., 1982.

[3] R. Jain and T. Binford, "Dialogue: Ignorance, myopia, and naivete in computer vision systems," *CVGIP*, vol. 53, no. 1, pp. 112–117, 1991.

[4] R. Lotufo and F. Zampirolli, "Fast multi-dimensional parallel Euclidean distance transform based on mathematical morphology," in *Proccedings of SIBGRAPI, XIV Brazilian Symposium on Computer Graphics and Image Processing*, pp. 100–105, IEEE Computer Society, 2001.

[5] H. Eggers, "Two fast Euclidean distance transformations in $\mathbb{Z}^2$ based on sufficient propagation," *Computer Vision and Image Understanding*, vol. 69, pp. 106–116, jan 1998.

[6] O. Cuisenaire and B. Macq, "Fast Euclidean distance transformation by propagation using multiple neighborhoods," *Computer Vision and Image Understanding*, vol. 76, no. 2, pp. 163–172, 1999.

[7] T. Saito and J. Toriwaki, "New algorithms for Euclidean distance transformations of an n-dimensional digitised picture with applications," *Pattern Recognition*, vol. 27, no. 11, pp. 1551–1565, 1994.

[8] C. Maurer, R. Qi, and V. Raghavan, "A linear time algorithm for computing the Euclidean distance transform in arbitrary dimensions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 265–270, feb 2003.

[9] F. Y. Shih and Y.-T. Wu, "Fast Euclidean distance transformation in two scans using a 3×3 neighborhood," *Computer Vision and Image Understanding*, vol. 93, pp. 195 – 205, feb 2004.

[10] D. E. Knuth, "Big Omicron and big Omega and big Theta," *Sigact News*, vol. 8, no. 2, pp. 18–24, 1976.

[11] F. Aurenhammer, "Voronoi diagrams - a survey of a fundamental data structure," *ACM Computing Surveys*, vol. 23, no. 3, pp. 345–405, 1991.

[12] R. Fabbri, L. Estrozi, and L. Costa, "On Voronoi diagrams and medial axes," *Journal of Mathematical Imaging and Vision*, vol. 17, pp. 27–40, jul 2002.

[13] F. Preparata and M. Shamos, *Computational Geometry*. Springer-Verlag, 1990.

[14] P. Soille and L. Vincent, "Watersheds in digital spaces: An efficient algorithm based on imersion simmulations," *IEEE T. Patt. Anal. Mach. Intel.*, vol. 13, no. 6, pp. 583–598, 1991.

[15] I. Ragnemalm, "Neighborhoods for distance transformations using ordered propagation," *CVGIP - Image Understanding*, vol. 56, no. 3, pp. 399–409, 1992.

[16] R. Parker, *Algorithms for image processing and computer vision*. Wiley, 1997.

[17] M. Chen and P. Yan, "Multiscaling approach based on morphological filtering," *IEEE Patt. Anal. Mach. Intel.*, vol. 11, no. 7, pp. 694–700, 1989.

[18] Y. Ge and J. Fitzpatrick, "On the generation of skeletons from discrete Euclidean distance maps," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 11, pp. 1055–1066, 1996.

[19] P.-E. Danielsson, "Euclidean distance mapping," *Computer Graphics and Image Processing*, vol. 14, pp. 227–248, 1980.

[20] L. Vincent, "Exact Euclidean distance function by chain propagations," in *Proceedings of the IEEE Computer Vision and Pattern Recognition '91*, pp. 520–525, 1991.

[21] R. Coelho and L. Costa, "On the application of the Bouligand-Minkowski fractal dimension for shape characterisation," *Applied Signal Processing*, vol. 3, pp. 163–176, 1996.

[22] G. Borgefors and I. Nyström, "Efficient shape representation by minimizing the set of centres of maximal discs/spheres," *Pattern Recognition Letters*, pp. 465–472, 1997.

[23] Y. Chin, H. Wang, L. P. Tay, , and Y. C. Soh, "Vision guided AGV using distance transform," in *Proceedings of 32nd International Symposium on Robotics*, pp. 1416–1421, apr 2001.

[24] G. Borgefors, "Distance transformations in digital images," *Computer Vision, Graphics, and Image Processing*, vol. 34, pp. 344–371, 1986.

[25] D. W. Paglieroni, "Distance transforms: Properties and machine vision applications," *Graphical Models and Image Processing*, vol. 54, no. 1, pp. 56–74, 1992.

[26] H. Liu and M. Srinath, "Partial shape classification using contour matching in distance transformation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 11, pp. 1072–1079, 1990.

[27] J. You, E. Pissaloux, W. P. Zhu, and H. A. Cohen, "Efficient image matching: A hierarchical chamfer matching scheme via distributed system," *Real-time Imaging*, vol. 1, pp. 245–259, oct 1995.

[28] A. Rosenfeld and J. Pfaltz, "Distance functions on digital pictures," *Pattern Recognition*, vol. 1, no. 1, pp. 33–61, 1968.

[29] D. Kozinska, O. J. Tretiak, J. Nissanov, and C. Ozturk, "Multidimensional alignment using the Euclidean distance transform," *Graphical Models and Image Processing*, vol. 59, pp. 373–496, nov 1997.

[30] Y. hao Tseng, J. neng Hwang, and F. H. Sheehan, "3D heart modeling and motion estimation based on continuous distance transform neural networks and affine transform," *The Journal of VLSI Signal Processing–systems For Signal, Image, and Video Technology*, vol. 18, apr 1998.

[31] M. Wintermark, M. Reichhart, O. Cuisenaire, P. Maeder, J. Thiran, P. Schnyder, J. Bogousslavsky, and R. Meuli, "Comparison of admission perfusion computed tomography and qualitative diffusion- and perfusion-weighted magnetic resonance imaging in acute stroke patients," *Stroke*, vol. 8, pp. 2025–2031, aug 2002.

[32] F. Jolesz, W. Loresen, H. Shinmoto, H. Atsumi, S. Nakajima, P. Kavanaugh, P. Saiviroonporn, S. Seltzer, S. Silverman, M. Phillips, and R. Kikinis, "Interactive virtual endoscopy," *American Journal of Radiology*, vol. 169, pp. 1229–1237, 1997.

[33] J. Thiran and B. Macq, "Morphological feature extraction for the classification of digital images of cancerous tissues," *IEEE Transactions on Biomedical Engineering*, vol. 43, pp. 1011–1019, 1996.

[34] V. Starovoitov, "A clustering technique based on the distance transform," *Pattern Recognition Letters*, vol. 17, no. 3, pp. 231–239, 1996.

[35] P. Zeng and T. Hirata, "Distance map based enhancement for interpolated images," in *Geometry, Morphology and Computational Imaging*, vol. 2616 of *Lecture Notes in Computer Science*, pp. 86–100, Springer-Verlag, 2003.

[36] D. W. Paglieroni, "Directional distance transforms and height field preprocessing for efficient ray tracing," *Graphical Models and Image Processing*, vol. 59, pp. 253–264, jul 1997.

[37] A. J. Travis, D. J. Hirst, and A. Chesson, "Automatic classification of plant cells according to tissue type using anatomical features obtained by the distance transform," *Annals of Botany*, vol. 78, pp. 325–331, 1996.

[38] I. Kovács and B. Julesz, "Perceptual sensitivity maps within globally defined visual shapes," *Nature*, vol. 370, no. 6491, pp. 644–646, 1994.

[39] B. B. Kimia, "On the role of medial geometry in human vision," *Journal of Physiology*, 2003. Accepted for publication.

[40] H. Blum, *A transformation for extracting new descriptors of shape*, pp. 362–380. Models for the Perception of Speech and Visual Forms, Cambridge, Mass.: Mit Press, 1967.

[41] M. Leyton, *Symmetry, Causality, Mind*. Mit Press, apr 1992.

[42] G. Borgefors, "Distance transformations in arbitrary dimensions," *Computer Vision, Graphics, and Image Processing*, vol. 27, pp. 321–345, 1984.

[43] O. Cuisenaire, *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing*. PhD thesis, Université Catholique de Louvain, Belgique, oct 1999.

[44] L. Boxer and R. Miller, "Efficient computation of the Euclidean distance transform," *Computer Vision and Image Understanding Notes*, vol. 80, pp. 379–383, 2000.

[45] M. L. Gavrilova and M. H. Alsuwaiyel, "Two algorithms for computing the Euclidean distance transform," *International Journal of Image and Graphics*, vol. 1, no. 4, pp. 635–645, 2001.

[46] F. Shih and J. Liu, "Size-invariant four-scan Euclidean distance transformation," *Pattern Recognition*, vol. 31, no. 11, pp. 1761–1766, 1998.

[47] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman, "Linear time Euclidean distance transform algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 5, pp. 529–533, 1995.

[48] J. Mullikin, "The vector distance transform in two and three dimensions," *Graphical Models and Image Processing*, vol. 54, no. 6, pp. 526–535, 1992.

[49] R. Fabbri, "Animal – An IMAge Library." `http://animal.sourceforge.net`.

[50] R. Fabbri, "SIP – the Scilab Image Processing toolbox." `http://siptoolbox.sourceforge.net`.

[51] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*. Academic Press, 1976.

[52] A. Falcao, J. Stolfi, and R. A. Lotufo, "The image foresting transform: theory, algorithms, and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 19–29, jan 2004.

[53] E. O. Brigham, *The Fast Fourier Transform and Its Applications*. Prentice Hall, 1988.

[54] F. A. Zampirolli and R. A. Lotufo, "Classification of the distance transformation algorithms under the mathematical morphology approach," in *Proccedings of SIBGRAPI, XIII Brazilian Symposium on Computer Graphics and Image Processing*, (Gramado, RS, Brasil), 2000.

[55] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Math.*, vol. 1, pp. 269–271, 1959.

[56] E. F. Moore, "The shortest path through a maze," in *roc. Internat. Symp. Switching Th., Part II*, (Cambridge, MA), pp. 285–292, Harvard University Press, 1959.

[57] R. Bellman, "On a routing problem," *Quaterly of Applied Mathematics*, vol. 16, pp. 87–90, 1958.

[58] R. B. Dial, "Shortest path forest with topological ordering," *Communications of the ACM*, vol. 12, pp. 632–633, 1969.

[59] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, feb 1993.

[60] A. V. Aho, J. E. Hopcroft, and J. Ullman, *Data Structures and Algorithms*. Addison Wesley, 1982.

[61] U. Montanari, "A method for obtaining skeletons using a quasi-Euclidean distance," *Journal of the Association For Computing Machinery*, vol. 15, no. 4, pp. 600–624, 1968.

[62] B. Verwer, P. Verbeek, and S. Dekker, "An efficient uniform cost algorithm applied to distance transforms," *IEEE Transactions on Patt. Anal. and Mach. Intel.*, vol. 11, pp. 425–429, apr 1989.

[63] J. Piper and E. Granum, "Computing distance transforms in convex and non-convex domains," *Pattern Recognition*, vol. 20, no. 6, pp. 599–615, 1987.

[64] Y. Sharaiha and N. Christofides, "A graph-theoretic approach to distance transformations," *Pattern Recognition Letters*, vol. 15, pp. 1035–1041, 1994.

[65] M. Akmal Butt and P. Maragos, "Optimum design of chamfer distance transforms," *Image Processing, IEEE Transactions On Image Processing, IEEE Transactions On Image Processing, IEEE Transactions On*, vol. 7, oct 1998.

[66] F.leymarie and M.d.levine, "Fast raster scan distance propagation on the discrete rectangular lattice," *Computer Vision and Image Understanding*, vol. 55, jan 1992.

[67] I. Ragnemalm, "The Euclidean distance transformation in arbitrary dimensions," *Pattern Regocgnition Letters*, vol. 14, pp. 883–888, 1993.

[68] O. Cuisenaire and B. Macq, "Fast and exact signed Euclidean distance transformation with linear complexity," in *ICASSṔ99 - IEEE Intl Conference on Acoustics, Speech and Signal Processing*, vol. 6, (Phoenix, USA), pp. 3293–3296, mar 1999.

[69] D. W. Paglieroni, "A unified distance transform algorithm and architecture," *Mach. Vision Appl.*, vol. 5, no. 1, pp. 47–55, 1992.

[70] M. N. Kolountzakis and K. N. Kutulakos, "Fast computation of the Euclidean distance maps for binary images: Fast computations of the Euclidean distance maps for binary images," *Inf. Process. Lett.*, vol. 43, no. 4, pp. 181–184, 1992.

[71] L. Chen and H. Y. H. Chuang, "A fast algorithm for Euclidean distance maps of a 2-d binary image," *Information Processing Letters*, vol. 51, pp. 25–29, 1994.

[72] T. Hirata, "A unified linear-time algorithm for computing distance maps," *Inf. Process. Lett.*, vol. 58, no. 3, pp. 129–133, 1996.

[73] A. Meijster, J. Roerdink, and W. Hesselink, "A general algorithm for computing distance transforms in linear time," in *Mathematical Morphology and its Applications to Image and Signal Processing (Proc. 5th Int. Conf)*, 2000.

[74] R. A. Lotufo and F. A. Zampirolli, "Multidimensional parallel EDT using 1d erosions by propagation," march 2003. submetido.

[75] F. Y. C. Shih and O. R. Mitchell, "A mathematical morphology approach to Euclidean distance transformation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, pp. 197–204, 1992.

[76] C. Huang and O. Mitchell, "A Euclidean distance transform using grayscale morphology decomposition," *Pattern Analysis and Machine Intelligence, IEEE Transactions On*, vol. 16, pp. 443–448, apr 1994.

[77] F. A. Zampirolli, *Transformada de Distância por Morfologia Matemática.* PhD thesis, UNICAMP, Campinas, Brasil, jun 2003.

[78] S. J. Fortune, "A sweepline algorithm for Voronoi diagrams," *Algorithmica*, no. 2, pp. 153–174, 1987.

[79] R. L. Ogniewicz and O.Kübler, "Voronoi tessellation of points with integer coordinates: Time-efficient implementation and online edge-list generation," *Pattern Recognition, Pergamon Press*, vol. 28, 1995.

[80] W. Guan and S. Ma, "A list-processing approach to compute Voronoi diagrams and the Euclidean distance transform," *Pattern Analysis and Machine Intelligence*, vol. 20, pp. 757–761, jul 1998.

[81] J. Serra, *Image Analysis and Mathematical Morphology*, vol. 1. Academic Press, 1982.

[82] A. X. Falcao, L. F. Costa, and B. S. da Cunha, "Multiscale skeletons by image foresting transform and its application to neuromorphometry," *Pattern Recognition*, vol. 35, pp. 1571–1582, jul 2002.

[83] A. Biere, "The ccmalloc memory-leak tracing tool." `http://www2.inf.ethz.ch/personal/biere/projects/ccmalloc/#introduction`.

[84] J. Seward, "Valgrind: a GPL'd system for debugging and profiling x86-Linux programs." `http://valgrind.kde.org`.

[85] INRIA, "Scilab numerical programming environment." `www.scilab.org`.

[86] R. Fabbri, O. M. Bruno, L. da F. Costa, and J. C. Torelli, "Resultados completos e implementação de algoritmos relativos ao trabalho 'Comparação e desenvolvimento de algoritmos de transformada de distância euclidiana e aplicações'.." `http://cyvision.if.sc.usp.br/~rfabbri/edt`.

[87] J. Sethian, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry*. Cambridge University Press, 1999.

[88] R. Fabbri, L. F. Costa, and O. M. Bruno, "Scilab and SIP for image processing," *Linux Journal*, 2004. accepted.