UNIVERSIDADE DE SÃO PAULO
Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto
Departamento de Computação e Matemática

Gradient Estimation Using Wide Support Operators

Análise e Processamento de Imagens Médicas

Relatório do Projeto
Final de Disciplina

Alunos: Marcelo Gomes de Paula [5566380]
Rafael Delalibera Rodrigues [9596742]

Programa de Pós-Graduação em Computação Aplicada

Ribeirão Preto, SP
Novembro de 2015

0. Organização do Presente Trabalho

Este trabalho é parte integrante da disciplina de "Análise e Processamento de Imagens Médicas". Tendo por finalidade a escolha de um artigo para estudo. O artigo escolhido por nossa dupla foi "Gradient estimation using wide support operators" (do autor Senel, H. G.).

Nosso relatório está organizado de forma a apresentar uma introdução ao problema e motivação da pesquisa realizada no artigo. Assim o item de introdução é basicamente uma tradução com poucas alterações do conteúdo apresentado na introdução do artigo; seguindo sempre que possível a fidelidade da problemática apresentada pelo autor. Já o segundo item disserta sobre as bases teóricas utilizadas no artigo, tendo seu conteúdo apresentado como um resumo de nossa visão sobre os operadores apresentados; incluindo algumas imagens de outros artigos e com algumas modificações visando facilitar a compreensão dos conceitos abordados.

O terceiro item descreve algumas considerações sobre nossa implementação da metodologia apresentada no artigo. Lá são citados passos algoritmos e as decisões que tomamos na implementação do mesmo; vale citar que o artigo não estabelece critérios para a implementação dos algoritmos e nossa busca rápida na literatura também não nos forneceu uma orientação neste aspecto. Portanto, muitas das decisões e estratégias adotadas advêm das nossas experiências, podendo apresentar desempenho muito inferior aos algoritmos utilizados pelo autor.

Por fim, incluímos uma seção de conclusão que trata-se do resumo das conclusões do próprio artigo somadas de algumas considerações nossas a respeito dos resultados do nosso trabalho final da disciplina.

1. Introdução [Detecção de Bordas]

Detecção de bordas, simplificadamente, é o processo de localizar as transições de intensidade dos pixels de uma imagem. O sucesso na detecção de borda fornece uma boa base para o desempenho de tarefas de processamento de imagem de nível superior, tais como:

- Reconhecimento de objetos,
- Rastreamento de alvos,
- Segmentação.

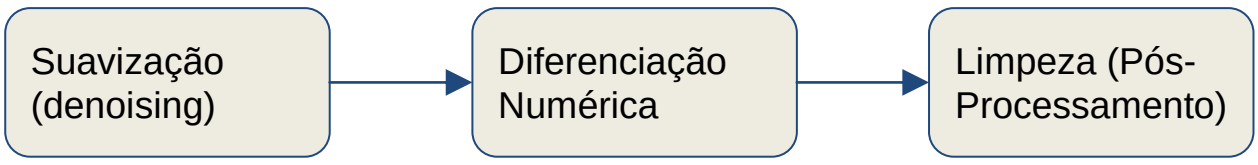
Os métodos mais rápidos de localização de bordas em imagens são aqueles baseados em máscaras pequenas (do inglês *kernels*), como **Sobel**, **Prewitt**, e **Roberts**. Porém, apesar destas máscaras pequenas fornecerem uma maneira rápida de computar os gradientes, elas têm pouco controle sobre o ruído, localização e orientação da borda; sendo sensíveis às transições do tipo degrau (do inglês *step edge*), mas não apresentando eficácia na detecção de transições suaves e remoção de ruídos (que acabam por serem incorretamente identificados como bordas).

Por outro lado, máscaras grandes fornecem características de supressão de ruído superior, mas sofrem com uma ampla área de resposta em torno de arestas, fazendo com que bordas dos objetos vizinhos possam se fundir, levando à identificação de apenas um objeto onde pode deveriam ser identificados dois ou até mais. Estes problemas associados com máscaras grandes de gradiente impedem a sua utilização generalizada.

O artigo apresenta um método que se utiliza de uma representação da imagem convertida para uma topologia *fuzzy* em conjunto com gradientes de máscaras grandes. O novo método limita eficazmente a área de resposta em torno da borda, evitando o comportamento indesejável de fusão de objetos vizinhos.

No artigo, o método é aplicado a imagens sintéticas, com o intuito de demonstrar a supressão de ruído superior. Imagens naturais também são utilizadas para avaliar o desempenho da estimativa proporcionada pela mascará.

Ao longo dos anos, foram propostas muitas técnicas para detecção de bordas, porém a maioria consiste nos três passos seguintes:



Devido à natureza altamente sensível a ruídos, uma etapa de alisamento ou *denoising* pode ser (e na grande maioria das vezes será) necessária uma vez que as imagens adquiridas por um sensor de imagem são muitas vezes degradadas pelo ruído. As imagens são suavizadas usando algum filtro passa-baixa (do inglês *low-pass filter*), por exemplo, o filtro Gaussiano. Então pequenas máscaras de gradiente (de dimensões 3x3 ou 5x5) são aplicadas; esta etapa é denominada diferenciação numérica. Em seguida, é aplicado um limiar (*threshold*) ao gradiente de saída e técnicas de pós-processamento são usadas para eliminar falsos positivos, a fim de formar um mapa de borda binário.

Há quatro fatores que afetam o desempenho dos esquemas de detecção de borda com base no uso de gradientes:

1) Grau de suavização e remoção de ruído,
2) Precisão da diferenciação numérica,
3) Modelo de borda e ruído utilizada no projeto do filtro,
4) Eficiência do pós-processamento.

A supressão de ruído é um passo necessário, uma vez que a operação de diferenciação é altamente sensível ao ruído, especialmente para ruídos de alta frequência (que irão afetar muito o desempenho do método de supressão). Por outro lado, aplicar uma suavização pesada pode remover alguns dos detalhes finos. Por consequência desta problemática, em algumas aplicações, o grau de suavização pode ser insuficiente para remover o ruído de forma eficaz.

Operadores de **Sobel** e **Prewitt** são os métodos de estimativa de gradiente mais simples disponíveis na. Estes se tornaram os operadores mais utilizados devido à sua simplicidade computacional. No entanto, são discutíveis no que se refere à sua precisão no cálculo das diferenças.

Neste contexto, ao comparar as respostas dos operadores de **Sobel** e **Prewitt** com a versão discreta do gradiente contínuo ideal, **Ando** descobriu que estes operadores não calculam o gradiente verdadeiro; tendo pouco controle sobre o ruído, localização e orientação borda. Eles podem detectar somente bordas com determinadas orientações e apresentam um desempenho fraco quando essas bordas são turvas e contaminadas com ruído. O seu tamanho da máscara e seus coeficientes fixos não

permitem que este gradiente se adapte a uma dada imagem.

Detectores de bordas normalmente são concebidos para serem sensíveis a certos tipos específicos de bordas. A abordagem básica para a derivação de máscaras pequenas de gradiente é o método de ajuste à superfície, no qual se encaixa a superfície para os dados de uma imagem (numa certa janela). O modelo de borda está intimamente relacionado com o tamanho da máscara. Pequenos operadores (por exemplo, 3x3 5x5) asseguram que apenas uma borda está contida dentro das suas fronteiras. Já operadores maiores são necessários para modelos que contêm bordas mais largas.

Conforme observação de **Canny**, semelhante à do autor do artigo, não houve trabalho relacionada à detecção de borda que oferecesse suporte a múltiplos tipos de bordas em um único operador. Operadores com menor comprimento são aconselhados sempre que a relação sinal-para-ruído (SNR) é suficiente a fim de evitar a (possibilidade de) interferência de bordas vizinhas.

O modelo de borda baseado em degrau é um passo muito restritivo por duas razões. Em primeiro lugar, a maioria das imagens contêm diferentes estruturas suavizadas além das bordas abruptas (degrau); imagens naturais são conhecidas por terem propriedades estatísticas que muitas vezes se contradizem com a suposição dessas bordas do tipo degrau. Em segundo lugar, objetos vizinhos na cena podem apresentar um impacto considerável sobre o resultado de detecção de borda, especialmente para máscaras grandes.

Ao alterarmos o modelo de borda para capturar bordas com transição lenta (suave), por exemplo, a borda da rampa, esta tarefa se torna um problema com os métodos baseados em gradiente com máscara pequena. Pois muitas bordas suaves ultrapassam a janela do operador e muitos componentes suavizados internos a essa janela não são bordas, mas componentes de ruído.

Para determinar um nível limite global ótimo para o gradiente das imagens, **Henstock** propôs um método com base em modelo estatístico. Um trabalho semelhante visa encontrar um limite para as saídas num detector de borda de **Canny**. A aplicabilidade destes métodos para todos os tipos de borda ou imagens de gradiente é discutível já que as propriedades estatísticas locais podem variar significativamente nas imagens (em contraposição as suas propriedades globais).

Ao considerar a dificuldade de determinar um limiar único, **Aggoun** e

Khallil propuseram um método com base em histogramas locais de pequenos blocos que não se sobrepõem no mapa de bordas. A fim de minimizar o efeito da escala de suavização, vários tamanhos de blocos são usados e os seus resultados são combinados para formar uma imagem de borda binária.

Com a introdução de um limiar histerese e supressão de falso positivo, **Canny** propôs um método baseado em gradiente de detecção de borda ideal. O filtro ideal é concebido através da maximização de três critérios: a relação sinal-ruído da imagem (SNR), a precisão da localização de borda, e uma resposta para cada borda. O modelo de borda degrau é assumido e a precisão de localização é alcançada usando o operador gradiente de primeira ordem. É, então, usado o pós-processamento para eliminar falsas bordas verificando vizinhos de cada pixel. O problema com a detecção de borda de **Canny** é que o algoritmo marca um ponto como uma borda caso sua amplitude seja maior do que a dos seus vizinhos sem verificar a possibilidade de ruído aleatório. Sua técnica é, então, susceptível a bordas inexistentes e instáveis que aparecem na vizinhança de objetos suavemente sombreados e fronteiras borradas. Esse comportamento não surpreende, pois o filtro ideal usa um pequeno número de pixels e sua máscara não é grande o suficiente para examinar toda a extensão de bordas suavizadas e largas.

Estes problemas estão claramente associados com a pressuposição de uma borda do tipo degrau e do uso de um número limitado de pixels vizinhos; tanto para o cálculo do gradiente, quanto para utilização nos passos de supressão de ruídos.

Detecção de bordas é um problema mal colocado. É difícil conceber um detector de bordas que localize todas as bordas verdadeiras com suas diferentes características, sem detectar outras estruturas. O sucesso de um detector de bordas depende dos resultados de sua etapa de suavização (remoção de ruídos) e da fase de estimativa do gradiente. A maioria das técnicas de pós-processamento utilizadas em esquemas de detecção de borda procuram eliminar problemas específicos causados em etapas anteriores. Em vez da concepção de melhores métodos de pós-processamento, a estimativa do gradiente e suavização podem ser melhoradas através do aumento do tamanho do operador de gradiente.

Embora os operadores pequenos, comumente usados, sejam os melhores para descrever texturas detalhadas, os operadores grandes são conhecidos para relatar respostas de baixa amplitude mais confiáveis.

Com o aumento das resoluções das imagens, a necessidade de máscaras maiores cresce, a fim de realizar a detecção de borda em imagens de alta resolução. Operadores grandes, no entanto, têm problemas que têm impedido a sua utilização generalizada, como por exemplo, os elevados requisitos computacionais.

O objetivo do artigo é aprimorar o passo de estimativa do gradiente que existe na maioria dos algoritmos de detecção de borda usando operadores de suporte a estes operadores grandes. O método proposto pode ser integrado em qualquer algoritmo de detecção de borda à base de gradiente. No entanto, o grau de melhoria obtido por diferentes técnicas de detecção de borda está fora do escopo do trabalho do autor (sendo elencado como trabalho para futuras pesquisas).

O artigo centra-se na computação do gradiente de intensidade de imagens que requerem menos pós-processamento, fornecendo boa localização para as bordas e lidando com diferentes tipos de bordas. A fim de mostrar a superioridade do método proposto, um esquema de detecção de borda simples composto pela estimativa de gradiente e aplicação de um limiar é adoptada.

Suavização (por exemplo, Gaussiana) e métodos de pós-processamento não são usados.

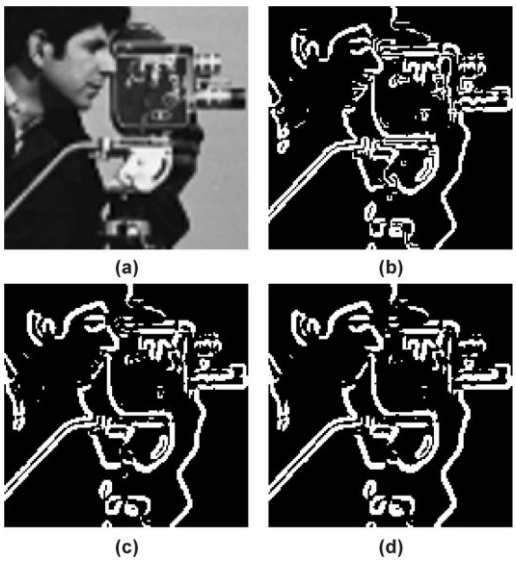


Figura 1 – Cinegrafista Máscaras [1]

A fim de mostrar a resposta de máscaras maiores, máscara com suportes 3x3, 7x7 e 9x9 dadas por **Farid** e **Simoncelli** são usadas para filtrar a imagem do cinegrafista (512x512).

(a) Imagem do cinegrafista (original)
(b) Saída com uma máscara 3x3
(c) Saída com uma máscara 7x7
(d) Saída com uma máscara 9x9

Para máscaras grandes, além de métodos adaptativos, a redução da espessura das linhas do gradiente pode ser conseguida, simplesmente, através do aumento do nível de limiar (threshold).

A figura a seguir mostra o mapa de borda binário de imagens do cinegrafista que são obtidas limitando a imagem gerada pelo gradiente em vários níveis.

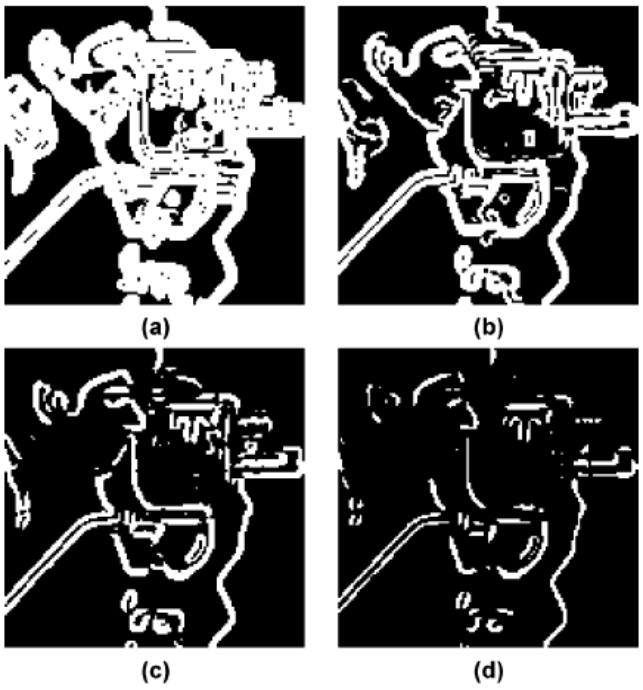


Figura 2 – Cinegrafista Limiares [1]

A imagem do cinegrafista é filtrada com uma mascara 9x9 e vários valores de limiar (*threshold*) m, 3 x m, 5 x m e 7 x m.

(a) m
(b) 3 x m
(c) 5 x m
(d) 7 x m

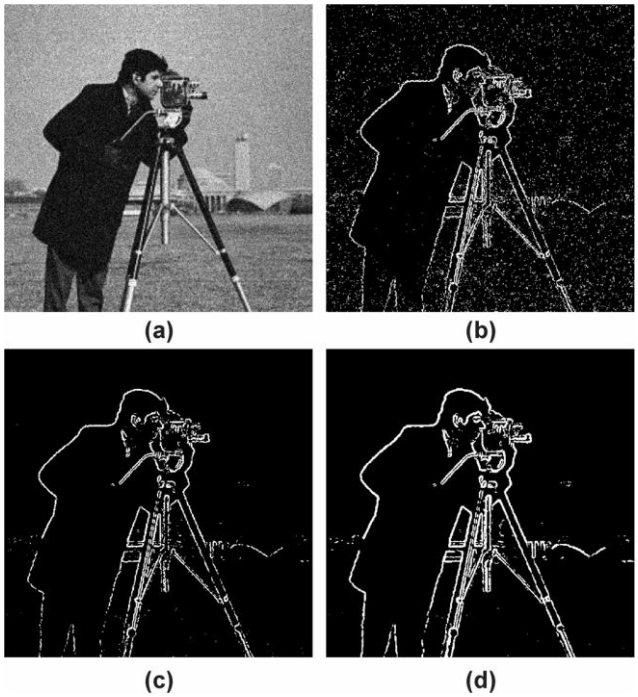


Figura 3 - Cinegrafista Ruído [1]

Para uma imagem do cinegrafista contaminada com ruído Gaussiano:

(a) Ruído Gaussiano;
(b) Saída 3x3;
(c) Saída 5x5;
(d) Saída 9x9.

O objetivo deste trabalho é encontrar formas de superar os problemas associados com máscaras grandes. Na busca de um esquema melhor para detecção de bordas baseado em gradiente, os seguintes critérios são levados em consideração:

1) O gradiente de saída não deve exceder a área da borda, onde existe a variação de intensidade de pixel.
2) O efeito da etapa de limiar deve ser mínimo.
3) Não pode haver várias bordas dentro dos limites de uma máscara, mas a interferência das bordas vizinhas deve ser eliminada.
4) A diferenciação numérica deverá ser o mais próximo possível da derivada ideal.
5) Deve ser atingido um bom nível de imunidade ao ruído para partes planas de imagens.

2. Bases Teóricas – Topologia Fuzzy

O método proposto pelo autor se utiliza do conceito de topologia fuzzy. Sendo necessário, então, transformar a imagem para um domínio topológico fuzzy, onde será possível utilizar operadores específicos deste domínio de representação e que terão grande utilidade no método proposto.

O conceito de topologia fuzzy tem sua origem na topologia digital, que se aplica às imagens binárias; permitindo a extração de diversas unidades de informação de grande valor ao processamento destas. Essa técnica é de grande valor por permitir diversas análises de grupos de conectividade presentes nas imagens. Porém, as técnicas de topologia digital não se aplicam de imediato às imagens em escala de cinza, por trabalharem sobre conjuntos binários. Com isso **Rosenfeld** (em 1979) introduziu uma extensão topológica para imagens multivaloradas (em escala de cinza) com a aplicação de uma topologia (modelagem) fuzzy sobre o conjunto de pixels dessas imagens.

Nessa topologia cada pixel tem seus valores mapeados para um intervalo entre 0 e 1, onde será então possível calcular graus de pertinência para estes pixels. O que então nos permitirá também utilizar operadores que permitam extrair informações relativas aos grupos de conectividade presentes nessa imagem.

A primeira noção que deve ser introduzida para utilização dessa topologia é o DOM (*Degree of Membership*; ou, em português, grau de pertinência), que se trata

basicamente da função de conversão dos valores de intensidade dos pixels para o domínio fuzzy. Essa pertinência pode ser calculada a partir de duas perspectivas complementares, denominadas por domínios de pertinência; o domínio dos pixels brilhantes e o domínio dos pixels escuros (referenciadas na literatura por *bright* e *dark*; *b* e *d*). O DOM é calculado através da função de pertinência fuzzy (do inglês *fuzzy membership function*) para o domínio em questão:

Para o domínio de pixels brilhantes:

$$\mu_I\,(p) = \frac{I(p)}{k};\;\; k = max\{I(p) \in \Sigma\}$$

Cujos parâmetros são:

- p: Pixel em questão.
- Σ: Conjunto de pixels/pontos da imagem.
- I: Matriz de intensidades dos pixels.
- $I(p)$: Matriz de intensidades calculada em p.
- $\mu_I(p)$: Função de pertinência para o ponto p.
- k: Valor de maior intensidade no conjunto de pixels.

Ou seja, o DOM de um pixel é sua intensidade normalizada para o conjunto de valores no intervalo [0,1]. Como o domínio dos pixels escuros é complementar ao domínio dos pixels brilhantes, ele pode ser calculado da seguinte maneira:

$$\mu_{\bar{I}}\,(p) = 1 - \mu_I\,(p)$$

Vale notar que quando não explicitado o domínio, este normalmente deverá ser interpretado como o domínio dos pixels brilhantes. Porém, o domínio utilizado normalmente será identificado pela presença do índice *b* ou *d* (exemplo: DOM_b ou DOM_d).

Após a definição do DOM e consequente transformação da imagem para um domínio fuzzy. É possível definir o operador DOC (*Degree of Connectedness*; ou, em português, grau de conectividade). Para entender a função deste operador, basta nos

remetermos às topologias em imagens binárias, onde a noção de conectividade se resume a um pixel estar ou não conectado a outro. Porém essa noção não pode ser estabelecida diretamente no domínio fuzzy, onde surge uma noção de grau de conectividade (DOC), ou seja, quão conectado certo pixel está com relação a outro.

Para definição então do grau de conectividade, precisamos antecipadamente definir a noção de força de um caminho entre dois pixels (p e q).

Seja **ρ** um caminho qualquer entre **p** e **q**:

$$\boldsymbol{\rho} = \{p = p_0, p_1, \dots, p_n = q\} \subset E \subset \Sigma$$

A força (do inglês, *strength*) deste caminho será dada por:

$$s_E\,(\boldsymbol{\rho}) \equiv \min_{0 \le i \le n} \{\mu_I\,(p_i)\}$$

Ou seja, um caminho é tão forte quanto seu elo mais fraco; seu pixels com menor grau de pertinência (DOM).

Como exemplo, podemos observar a figura abaixo, onde dois possíveis caminhos entre **p** e **q** são representados. Vemos que um dos caminhos (à direita) passa por um pixel **r** de baixíssima intensidade, sendo este o pixel com valor mínimo para o grau de pertinência entre todos os pixels do caminho; determinando, então, a força do caminho para o seu valor de grau de pertinência. Enquanto que aquele caminho mais à esquerda tem sua força determinada pelos pixels de intensidade média localizados na proximidade do ponto **q**. Sendo então o caminho da esquerda mais forte que o da direita.

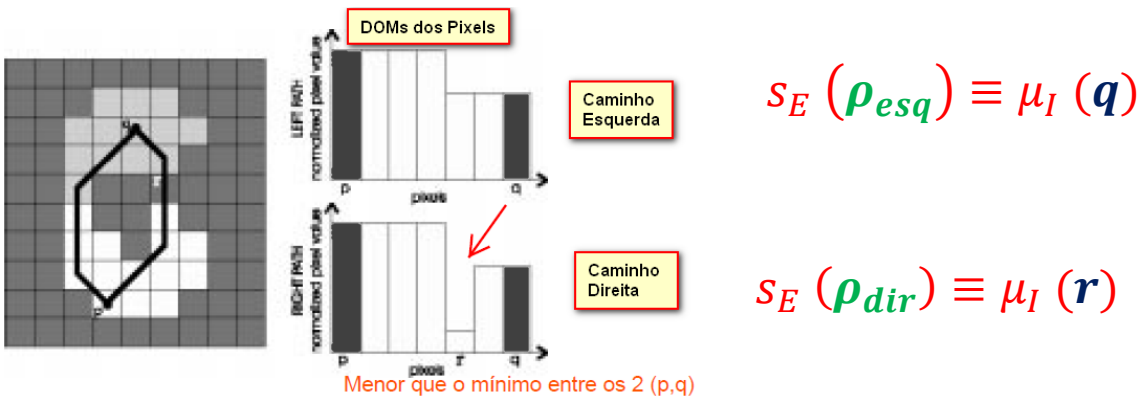


Figura 4 - Caminhos entre dois pontos e seus DOMs. Adaptado de [2]

Esta noção é importante, pois, com ela, podemos definir o grau de conectividade entre dois pontos, que será dado pelo caminho entre os dois pontos que possua a força máxima (entre todos os caminhos possíveis num determinado espaço E, onde $E \subset \Sigma$).

Grau de conectividade entre 2 pontos (DOC) é definido, então, como:

$$c_E\,(p,q) \equiv \max_{\rho \subset E}\{s_E\,(\boldsymbol{\rho})\}$$

O subconjunto E pode ser interpretado como uma janela de observação, com a finalidade de proporcionar um ambiente computável e maior controle sobre as dimensões de conectividade que se deseje analisar.

Dadas estas definições, dizemos que dois pixels estão conectados em E, quando:

$$\mu_I\,(p_i) \geq min\{\mu_I\,(p), \mu_I\,(q)\} \text{ ou,}$$
$$DOC = c_E\,(p,q) = min\{\mu_I\,(p), \mu_I\,(q)\}$$

Para o caminho:

$$\rho^c = \{p = p_0, p_1, \ldots, p_n = q\}; p_i \in E$$

Ou seja, dois pixels são considerados conectados quando há pelo menos um caminho entre eles cujos pixels componentes não tenham intensidade inferior ao ponto de saída e chegada. Em outras palavras, não pode haver um pixel do caminho com intensidade menor que o mínimo da intensidade dos pixels de saída e chegada (**p** e **q**). A imagem abaixo representa dois caminhos entre p e q, um que dizemos conectar estes pontos (à esquerda) e outro que assumimos não ser um caminho que conecte estes pontos (à direta); veja o ponto que impede esta conexão destacado em vermelho no gráfico de intensidades dos pontos deste caminho.

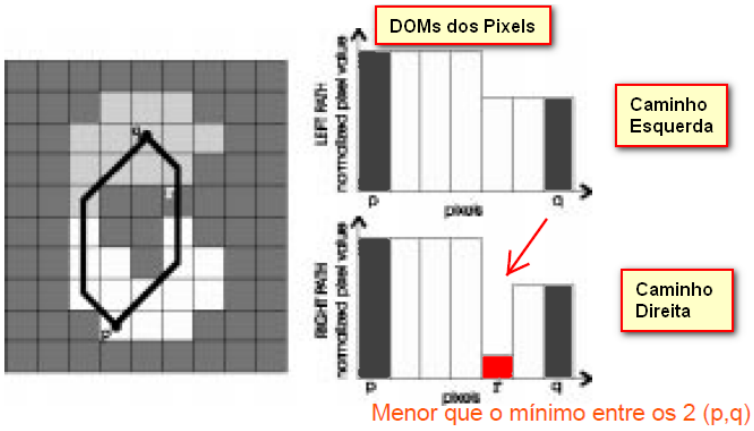


Figura 5 - Conectividade entre dois pontos. Adaptado de [2]

Pode-se então definir um novo operador denominado DOCM (*Degree of Connectedness Map*; ou, em português, mapa do grau de conectividade) numa janela fixa de observação, denominada comumente por **W** (análoga ao subconjunto **E**). O qual é formalmente definido como:

$$DOCM(p) = C(o,p);\ \forall p \in W$$

Onde **o** é o ponto/pixel de origem da observação (normalmente um ponto central numa janela quadrada de largura ímpar). Ou seja, trata-se de uma estrutura gerada a partir de uma janela de observação, onde os valores dos pixels em W são substituídos pelos respectivos **DOC** com relação a uma origem em **o**. Este mapeamento nos dá então para cada ponto com relação ao ponto de origem qual o grau máximo de conectividade entre esses dois pontos.

A análise dos resultados obtidos a partir de um operador DOCM, nos dão informações relevantes à respeito da estrutura da imagem analisada. Na figura abaixo podemos ver como se comporta a conectividade entre dois grupos de pixels separados por, digamos, uma vale de baixa intensidade com relação ao DOC. Interprete a figura abaixo como a representação das intensidades dos pixels através de uma única coluna da imagem, que foi extraída perpendicularmente a duas linhas borradas (visíveis na imagem e denominadas na figura por Linha 1 e Linha 2).

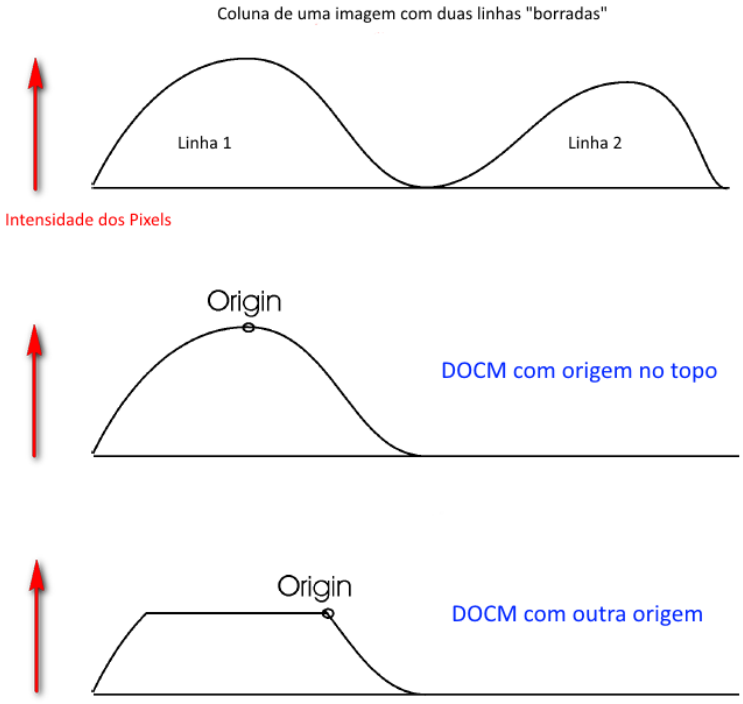


Figura 6 - Intensidades em duas linhas borradas. Adaptado de [1]

Um ponto da linha 1 não se conecta à linha 2, pois o valor de intensidade na origem e destino são maiores que o ponto no vale; o que leva os valores de força de todos os caminhos da linha 1 até a linha dois nesta janela de visualização para um valor de intensidade menor que o mínimo entre a origem e o destino. A figura também nos mostra que quando o ponto de origem não se encontra no pico de intensidade, as conexão são estabelecidas com base no limiar estabelecido pelo ponto de origem, ou seja, (relembrando) a força das conexões se dá sempre pelo menor valor de intensidade do caminho (que neste caso está limitado pela própria origem).

A figura abaixo mostra o comportamento do DOCM em diversas situações de interesse. À esquerda estão representadas transições do tipo degrau (*step edge*) e o comportamento diferenciado do operador DOCM quando da origem se encontrar no pixel mais brilhante ou no pixel mais escuro. Já na porção superior direita da imagem está representado o comportamento do DOCM em uma região de transição suave. E na porção inferior direita é representado o DOCM em uma região onde há a presença de um objeto vizinho. Em resumo a figura nos mostra que o DOCM é capaz de diferenciar os comportamentos em diversas situações.

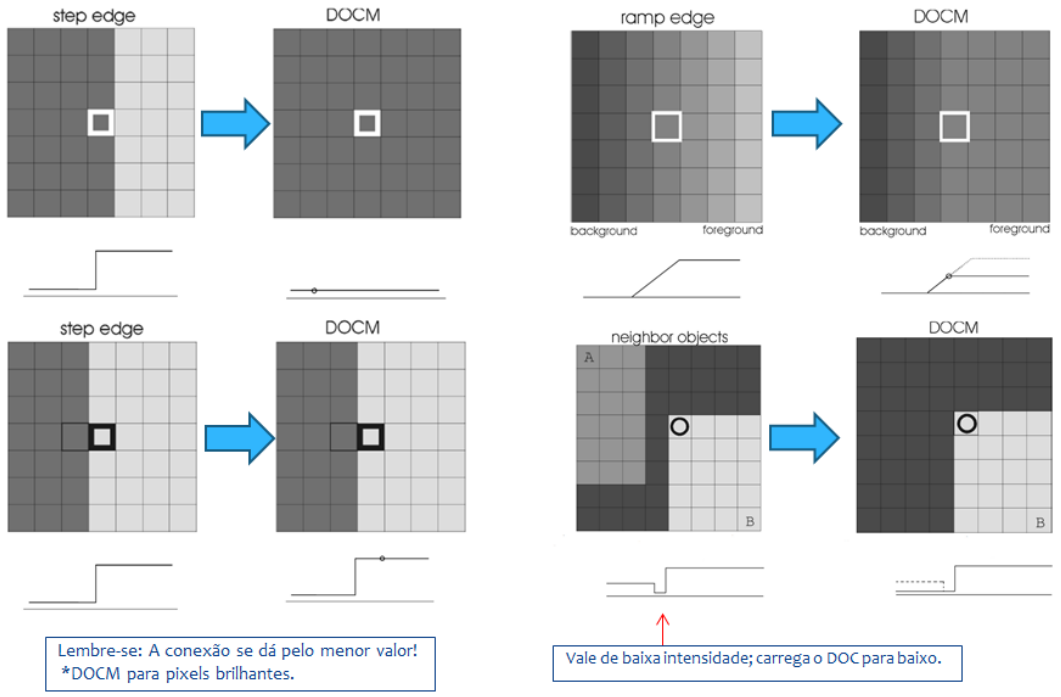


Figura 7 - DOCM em regiões diversas. Adaptado de [1,3]

O DOCM também apresenta uma característica interessante, podendo atuar como filtro de ruídos, conforme ilustrado na figura abaixo; onde são mostradas as características filtrantes de um operador DOCM no domínio dos pixels brilhantes (esquerda) e outro no domínio dos pixels escuros (direita).

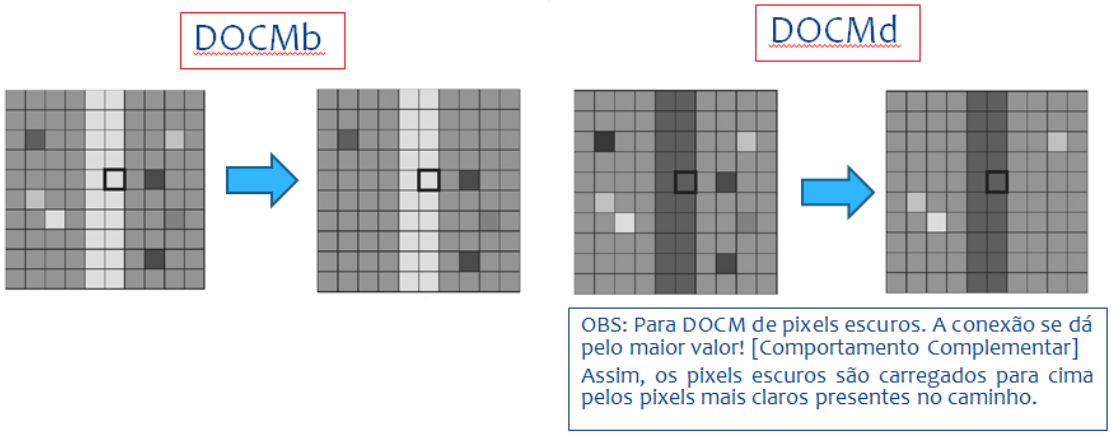


Figura 8 - DOCM como filtro de ruídos. Adaptado de [1]

Como é possível observar o operador aplicado no domínio dos pixels brilhantes acabou por remover ruídos mais brilhantes (dado que estes não formavam grupos de conexão com o ponto de origem da observação). Enquanto que o operador aplicado ao domínio complementar (dos pixels escuros) removeu os ruídos mais escuros (que também não apresentavam conectividade com o ponto de origem da observação). Porém, podemos deduzir que essas características não são observadas em janelas pequenas; já que numa janela 3x3 (como exemplo) todos os pixels estariam conectados. Portanto, para tirarmos proveito desta característica se faz necessário trabalharmos com janelas maiores que 5x5.

Dadas estas características assimétricas, porém complementares, observadas nos DOCM brilhante e escuro, é proposto pelo artigo um gradiente topológico fundamentado na combinação dos DOCM (em ambos os domínios) unidos ao gradiente topológico tradicional.

Seja então o DOCM formalizado da seguinte maneira:

$$W: (2n+1) \times (2n+1)$$

$$DOCMb(x, y; x_0, y_0) = \begin{cases} C\big(p(x,y), p(x_0,y_0)\big) \\ 0 \end{cases}$$

$$x_0 \in [x-n, x+n];$$
$$y_0 \in [y-n, y+n]$$

Onde W é a janela de observação para esse mapeamento, (x,y) são as coordenadas do ponto de origem da observação (ponto central da janela) e (x_0, y_0) são os outros pontos que compõem a janela. Assim, o DOCM será o cálculo do DOC ponto a ponto nessa janela (com origem sempre no centro) e será igual a 0 para qualquer ponto externo à W.

Teremos então quatro operadores topológicos direcionais:

- Dois gradientes topológicos através do eixo x:

$$TGb_x(x,y) = \sum_{\forall x_1,y_1} DOCMb(x,y;x_1,y_1)g_x(x_1,y_1)$$

$$TGd_x(x,y) = \sum_{\forall x_1,y_1} DOCMd(x,y;x_1,y_1)g_x(x_1,y_1)$$

- E dois gradientes topológicos através do eixo y:

$$TGb_y(x,y) = \sum_{\forall x_1,y_1} DOCMb(x,y;x_1,y_1)g_y(x_1,y_1)$$

$$TGd_x(x,y) = \sum_{\forall x_1,y_1} DOCMd(x,y;x_1,y_1)g_y(x_1,y_1)$$

Onde g_x e g_y são operadores convencionais de gradiente, com janela definida como:

$$w_g: (2k+1) \times (2k+1);\ \ k \leq n$$

$$\boxed{\begin{array}{c} x_1 \in [x-k, x+k]; \\ y_1 \in [y-k, y+k] \end{array}}$$

Ou seja, o operador de gradiente será utilizado dentro do mapa de conectividade (DOCM). As magnitudes dos gradientes topológicos são calculadas da seguinte forma:

$$TGb(x,y) = \sqrt{TGb_x(x,y)TGb_x(x,y) + TGb_y(x,y)TGb_y(x,y)}$$

$$TGd(x,y) = \sqrt{TGd_x(x,y)TGd_x(x,y) + TGd_y(x,y)TGd_y(x,y)}$$

Uma das premissas propostas pelo autor era que a área de resposta do gradiente deva compreender somente a seção onde ocorre a variação de intensidade.

Evitando assim o efeito conhecido como *edge blur* (borrão na borda). Como pode ser visto na figura abaixo os gradientes topológicos direcionais cumprem parcialmente este requisito; o que é avaliado positivamente, já que no caso de aplicação do gradiente convencional há a invasão dos limites da borda nas duas direções.

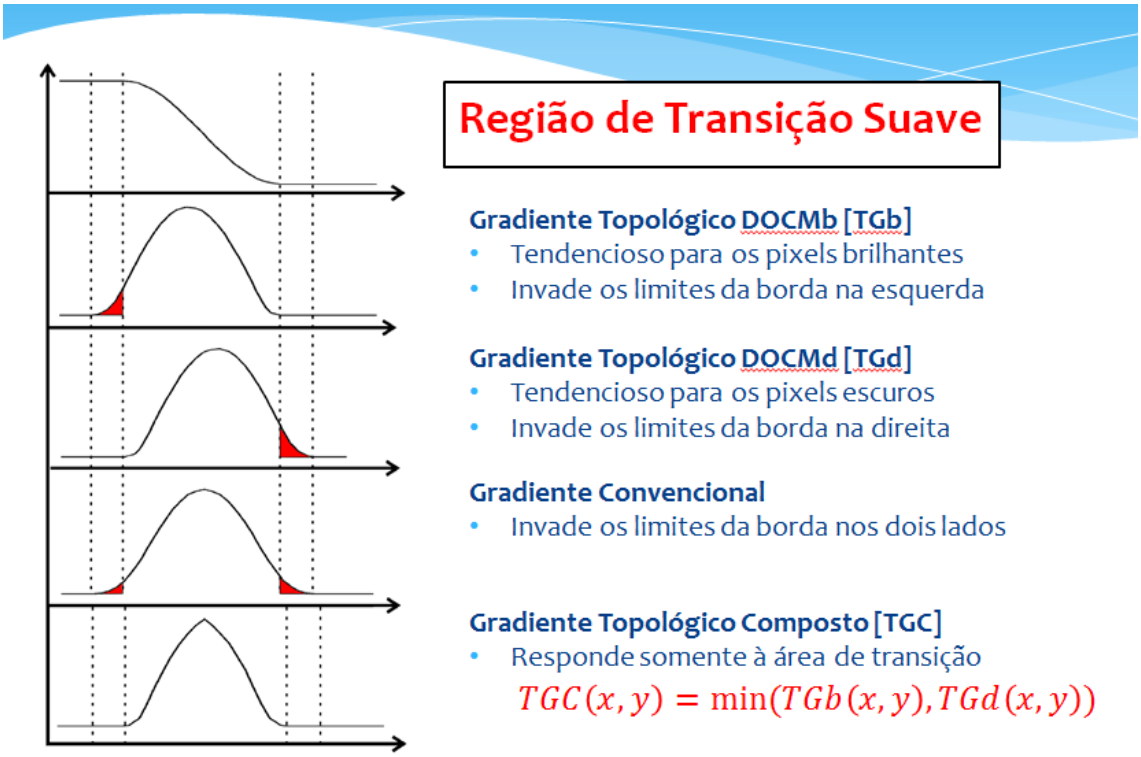


Figura 9 - Comportamento dos gradientes topológicos em região de transição suave. Adaptado de [1]

Então é definido o gradiente topológico composto (*Composite Topological Gradient*) denominado por TGC; que cumpre este requisito completamente. E é nada mais do que do que o mínimo entre os valores de TGb e TGd. Porém, ao analisarmos o comportamento deste gradiente topológico composto nos diferentes tipos de bordas previstos é possível identificar um problema para a borda do tipo degrau. Pois, dado que a magnitude de resposta do TGC é calculada com base nos valores mínimos, conforme uma transição se torna mais fina (abrupta) a magnitude de resposta assume valores cada vez menores. O que irá culminar em uma resposta quase nula quando da ocorrência de uma transição do tipo degrau. Este problema está ilustrado na figura abaixo, onde à esquerda são apresentados os componentes TGb e TGd do gradiente na ocorrência deste tipo de transição e à direita destaca-se em vermelho a resposta obtida no TGC; sendo insuficiente para identifica corretamente a transição.

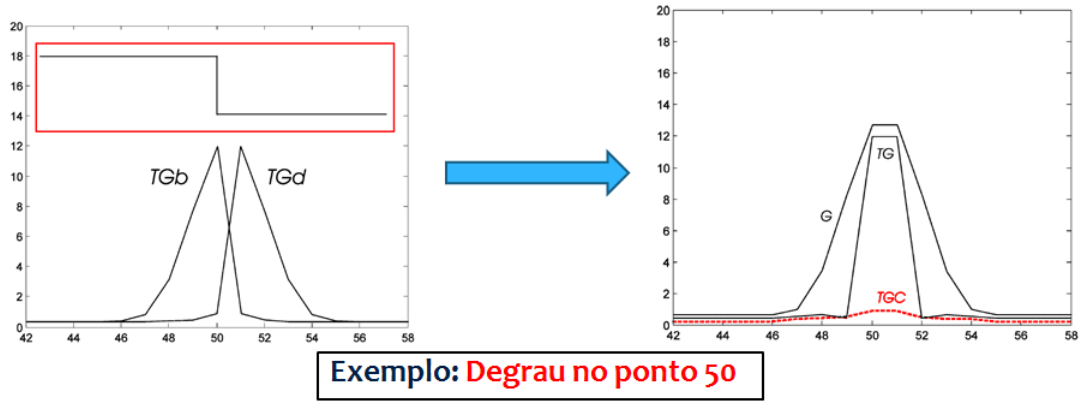


Figura 10 - Análise do TGC em região de transição do tipo degrau. Adaptado de [1]

Para solucionar este problema é necessário estabelecer uma condição diferenciada para o cálculo do TGC, na região de deslocamento do ponto mínimo. Ao incorporar essa restrição o gradiente passará a ser denominado por TG (gradiente topológico); por ser o gradiente topológico final proposto pelo artigo. A condição que deverá ser adicionada é que nesta região onde ocorre a transição abrupta o gradiente topológico deverá ser calculado pelo máximo entre seus componentes e não o mínimo, como ocorre em todas as outras regiões.

$$TG = \begin{cases} \max\big(TGb(x,y), TGd(x,y)\big), & \text{Na região de deslocamento do ponto mínimo} \\ \min\big(TGb(x,y), TGd(x,y)\big), & \text{Todo o resto} \end{cases}$$

Onde a região de deslocamento do ponto de mínimo é aquela cercada por dois mínimos seguidos, vindos de gradientes topológicos direcionais distintos:

- (x-1,y) é mínimo de TGd;
- (x+1,y) é mínimo de TGb.

Os mínimos podem ocorrer na contrária, o anterior em TGb e o sucessor em TGd. Vale observar que esta região é determinada de maneira análoga para o eixo y. Com essa última consideração o modelo de gradiente topológico proposto pelo autor se encerra.

3. Nossa Implementação

Como foi citado no item 0 de nosso relatório, o artigo não aborda as especificidades relacionadas à implementação dos algoritmos que realizarão os operadores utilizados e propostos pelo autor. Realizamos uma busca rápida na literatura em busca de algoritmos relacionados a estes operadores ou ferramentas que os implementassem, porém não conseguimos encontrar muita informação que fosse útil para nortear o desenvolvimento destes. Neste cenário iniciamos a implementação dos operadores topológicos conforme nossa interpretação das definições teóricas dos mesmos e nossas decisões quanto aos recursos computacionais para solucionar as diversas etapas para construção destes.

Abaixo, segue a lista de operadores que deveríamos implementar afim de alcançar o gradiente topológico final proposto pelo autor:

- DOMb [Implementado]
- DOMd [Implementado]
- DOCb [Implementado]
- DOCd [Implementado]
- DOCMb [Implementado]
- DOCMd [Implementado]
- TGbx
- TGdx
- TGby
- TGdy
- TGC
- TG
- gx (Sobel 9x9) [Implementado]
- gy (Sobel 9x9) [Implementado]

Infelizmente, devido ao tempo disponível e às dificuldades encontradas para implementação dos operadores (dada a não disponibilização dos algoritmos), não conseguimos implementar toda a solução; foram concluídos apenas os itens acima

listados que estão seguidos da marcação “implementado”. A ideia inicial era produzir um *plugin* para o software ImageJ que disponibiliza-se tais operações, ponto que não foi alcançado. Porém, conseguimos implementar diversos operadores, incluindo o DOCM que dentre todos os operadores listados, é sem dúvidas aquele de maior complexidade computacional, no qual gastamos a maior parte do tempo que dedicamos ao projeto.

Realizamos as tarefas de desenvolvimento em duas frentes, uma integrada ao ImageJ; através da alteração de um *plugin* com código-fonte aberto para facilitar os testes com os métodos de DOM e com os gradientes convencionais necessários (onde escolhemos utilizar Sobel; o artigo deixava esta escolha livre). Para estes dois tipos de operadores conseguimos realizar os testes diretamente no ImageJ obtendo os resultados que esperávamos.

Outra frente de trabalho, consistiu em desenvolver uma classe que implementasse os métodos para cálculo dos DOCM a qual preferimos implementar de maneira independente e integrá-la ao *plugin* noutro momento, dada a sua maior complexidade. Para este operador, também realizamos testes bem sucedidos, porém com valores numéricos já que não possuíamos o suporte do ambiente do ImageJ.

Dadas as decisões realizadas para implementação do DOCM, gostaríamos de ressaltar algumas informações que consideramos relevantes. Seguem as tarefas algorítmicas que elencamos para o cálculo do DOCM:

- Para todos os pontos que compõem a janela de observação, devem ser executadas as seguintes tarefas:
    1. Percorrer todos os caminhos possíveis partindo do ponto central da janela até um determinado ponto.
    2. Para cada caminho que ligue estes dois pontos, devemos armazenar a força do caminho.
    3. Após percorrer todos os caminhos e calcular suas respectivas forças, devemos calcular o DOC entre esses pontos (que se dá pela força máxima).
    4. O valor do DOC entre os pontos será um valor da matriz do DOCM.

As estratégias e considerações para construção do algoritmo e estruturas de dados utilizadas para o cálculo do DOCM (no domínio dos pixels brilhantes) são listadas:

1. O percurso é viabilizado com a utilização de um algoritmo de caminhada recursiva e completa na janela fornecida (já representada no domínio Fuzzy), com saída a partir do ponto central da janela. Isso já resolve a regra de como percorrer todos os pontos na janela, porém sabemos que não se trata de uma solução ótima.
    a. Garantimos a não ocorrência de ciclos dentro do mapa de pontos com a utilização de uma pilha para armazenamento dos pontos já visitados no percurso em questão. Assim que haja um retorno a um nó anterior via recursão, é removido o último nó presente na pilha (permitindo que aquele ponto seja visitado por outro percurso).
2. Ao "caminhar" por um caminho qualquer deve guardar o valor mínimo do percurso; ou seja, manter sempre o valor de intensidade do menor pixel já visitado no caminho. Para isso utilizamos uma matriz auxiliar de mesma dimensão da janela, que mantêm o mínimo acumulado em cada ponto já visitado.
    a. Isso garante que ao realizarmos retornos a nós anteriores via recursão, consigamos saber qual era o mínimo acumulado até tal ponto. Retornando o valor do ponto “abandonado” para um valor sinalizado como máximo (para não interferir).
    b. Uma estrutura do tipo pilha poderia nos servir, porém esta não está preparada para cálculo do mínimo de maneira trivial; sendo então descartada.
3. A cada pixel visitado seu DOC (por aquele caminho) pode ser calculado como o menor dos valores entre sua própria intensidade e o menor valor guardado para aquele percurso.
    a. Para este problema, utilizamos uma matriz de listas (ArrayList). Onde guardamos para cada ponto o valor de seu DOC numa lista. Já que não nos importa qual o caminho, mas

apenas os valores obtidos para todos os valores possíveis.

4. A matriz de resultado contendo o DOCM é obtida ao final de todos os percursos possível para todos os pontos na janela, com o cálculo do valor máximo para cada lista referente a um ponto de destino (a origem é sempre o ponto central).

Essas foram as considerações que julgamos relevantes para a compreensão da construção do nosso DOCM. Vale lembrar que as considerações acima são apresentadas no domínio dos pixels brilhantes, para o calculo do DOCM no domínio dos pixels escuros basta a inversão de algumas tarefas. Mais informações podem ser obtidas com as visualização do nosso código implementado, que encontra-se totalmente documentado.

4. Conclusão

O artigo apresentou um novo método para estimar gradientes, visando alcançar as características de supressão de ruído das máscaras grandes, mas sem os efeitos colaterais indesejados (como a fusão de objetos vizinhos).

A principal contribuição deste trabalho é a introdução de um novo passo, que permite detectar os pontos conectados, habilitando o uso de grandes máscaras para detecção de bordas. O novo método tem características de supressão de ruído semelhantes aos operadores de gradiente convencionais de máscaras grandes. E os resultados alcançados pelo operador TG se mostraram superiores aos gerados por operadores de gradiente convencionais; conforme a análise do autor.

A desvantagem do método proposto é o elevado requisito computacional. Onde o método proposto se mostra aproximadamente 11 vezes mais lento do que o operador gradiente convencional, quando máscaras 13 x 13 são usadas.

Além das conclusões relativas ao método proposto, gostaríamos de acrescentar nossas conclusões relativas ao trabalho no projeto e estudo do artigo. Primeiramente, vale citar que ficamos plenamente convencidos das vantagens do método apresentado pelo autor, porém não tivemos sucesso ao procurar por maiores informações a respeito da implementação e realização do mesmo. Encontramos pouco literatura que relacionasse o uso de topologia fuzzy no âmbito da estimativa de

gradientes e detecção de bordas.

Porém, a necessidade de desenvolver os algoritmos com base em nossas decisões se mostrou uma atividade prazerosa, apesar da maior dificuldade. Isso se tornou ainda mais válido ao verificarmos que os operadores de DOCM se mostraram válidos para cálculos de matrizes numéricas de teste.

Mantemos ainda o interesse em finalizar o projeto deste *plugin* tão logo quanto possamos, apesar de não termos conseguido alcançar este resultado dentro do prazo para finalização do projeto da disciplina. Ao que tudo indica e conforme nossa percepção, a finalização deste *plugin* é uma tarefa relativamente simples e de duração média já que todos os operadores complexos foram implementados. Podemos então citar duas atividades faltantes: uma consiste em desenvolver os operadores restantes, que são compostos por aqueles que já construímos, e outra consiste em testar tais operadores quando disponibilizados como *plugin* no ImageJ.

5. Referências

[1] **Senel, HG. Gradient Estimation Using Wide Support Operators. IEEE Transactions on Image Processing. 2009 Apr;18(4):867–78.**

[2] Senel HG, Peters I R.A., Dawant B. Topological median filters. IEEE Transactions on Image Processing. 2002 Feb;11(2):89–104.

[3] Senel HG. Image Gradient Estimation with Wide Support Kernels. In: 4th International Conference on Information and Automation for Sustainability, 2008 ICIAFS 2008. 2008. p. 132–7.

Apêndice A - [DOM.java]

```
import ij.*;

public class DOM {

	static public ImageAccess gSobelX(ImageAccess input) {
		int nx = input.getWidth();
		int ny = input.getHeight();

		int tamGradiente = 9;

		// Janela
		double arr[][] = new double[tamGradiente][tamGradiente];
		// Kernel do Gradiente
		double kernelx[][] =  { {-4,-3,-2,-1, 0, 1, 2, 3, 4},
					{-5,-4,-3,-2, 0, 2, 3, 4, 5},
					{-6,-5,-4,-3, 0, 3, 4, 5, 6},
					{-7,-6,-5,-4, 0, 4, 5, 6, 7},
					{-8,-7,-6,-5, 0, 5, 6, 7, 8},
					{-7,-6,-5,-4, 0, 4, 5, 6, 7},
					{-6,-5,-4,-3, 0, 3, 4, 5, 6},
					{-5,-4,-3,-2, 0, 2, 3, 4, 5},
					{-4,-3,-2,-1, 0, 1, 2, 3, 4} };

		double pixel, cont;

		ImageAccess out = new ImageAccess(nx, ny);
		for (int x = 0; x < nx; x++) {
			for (int y = 0; y < ny; y++) {
				input.getNeighborhood(x, y, arr);

				pixel = 0.0;
				cont = 0;
				for (int i = 0; i<tamGradiente;i++)
				{
					for(int j = 0;j<tamGradiente;j++)
					{
						double value =
arr[i][j]*kernelx[i][j];
						pixel += value;
						if(value!=0)
						{
							cont++;
						}
					}
				}
				if(cont!=0)
				{
					pixel = pixel/cont;
				}

				out.putPixel(x, y, pixel);
			}
		}
		return out;
	}
```

```
	static public ImageAccess gSobelY(ImageAccess input) {
		int nx = input.getWidth();
		int ny = input.getHeight();

		int tamGradiente = 9;

		// Janela
		double arr[][] = new double[tamGradiente][tamGradiente];
		// Kernel do Gradiente
		double kernely[][] =  { {-4,-5,-6,-7,-8,-7,-6,-5,-4},
					{-3,-4,-5,-6,-7,-6,-5,-4,-3},
					{-2,-3,-4,-5,-6,-5,-4,-3,-2},
					{-1,-2,-3,-4,-5,-4,-3,-2,-1},
					{0 , 0, 0, 0, 0, 0, 0, 0, 0},
					{1 , 2, 3, 4, 5, 4, 3, 2, 1},
					{2 , 3, 4, 5, 6, 5, 4, 3, 2},
					{3 , 4, 5, 6, 7, 6, 5, 4, 3},
					{4 , 5, 6, 7, 8, 7, 6, 5, 4} };

		double pixel, cont;

		ImageAccess out = new ImageAccess(nx, ny);
		for (int x = 0; x < nx; x++) {
			for (int y = 0; y < ny; y++) {
				input.getNeighborhood(x, y, arr);

				pixel = 0.0;
				cont = 0;
				for (int i = 0; i<tamGradiente;i++)
				{
					for(int j = 0;j<tamGradiente;j++)
					{
						double value =
arr[i][j]*kernely[i][j];
						pixel += value;
						if(value!=0)
						{
							cont++;
						}
					}
				}
				if(cont!=0)
				{
					pixel = pixel/cont;
				}

				out.putPixel(x, y, pixel);
			}
		}
		return out;
	}

	static public ImageAccess gSobel(ImageAccess input) {
		int nx = input.getWidth();
		int ny = input.getHeight();

		ImageAccess outGx = new ImageAccess(nx, ny);
		ImageAccess outGy = new ImageAccess(nx, ny);
		ImageAccess result = new ImageAccess(nx, ny);
```

```
        outGx = gSobelX(input);
        outGy = gSobelY(input);

        outGx.pow(2.0);
        outGy.pow(2.0);
        result.add(outGx, outGy);
        result.sqrt();

        return result;
}

static public ImageAccess domBright(ImageAccess input)
{
        int nx = input.getWidth();
        int ny = input.getHeight();
        double max = input.getMaximum();
        ImageAccess domImage = new ImageAccess(nx, ny);

        double valueIn = 0.0;
        double valueOut = 0.0;

        for (int x=0; x<nx; x++)
        {
                for(int y=0; y<ny; y++)
                {
                        valueIn = input.getPixel(x,y);
                        valueOut = valueIn/max;
                        domImage.putPixel(x, y, valueOut);
                }
        }

        return domImage;
}

static public ImageAccess domDark(ImageAccess input)
{
        int nx = input.getWidth();
        int ny = input.getHeight();
        ImageAccess domBrightImage = new ImageAccess(nx, ny);
        ImageAccess domDarkImage = new ImageAccess(nx, ny);

        domBrightImage = domBright(input);

        double value = 0.0;

        for (int x=0; x<nx; x++)
        {
                for(int y=0; y<ny; y++)
                {
                        value = 1 - domBrightImage.getPixel(x,y);
                        domDarkImage.putPixel(x, y, value);
                }
        }

        return domDarkImage;
}

static public double domBright(ImageAccess input, int x, int y)
{
        double valueIn = input.getPixel(x, y);
```

```
            double valueOut = valueIn/input.getMaximum();

            return valueOut;
        }

        static public double domDark(ImageAccess input, int x, int y)
        {
            double valueIn = domBright(input,x,y);
            double valueOut = 1-valueIn;

            return valueOut;
        }

}
```

Apêndice B - [DOCM.java]

```
package docm;


import java.util.Stack;
import java.util.ArrayList;
import java.util.Collections; //Usada para ordenar o ArrayList


public class DOCM {

    public static double[][] DOCM(double[][] janela, int
dimensaoJanela, char DOCMtype)
    {
        // Janela DEVE ser ímpar
        int linhas = dimensaoJanela;
        int colunas = dimensaoJanela;

        int[] qtdPassos = new int[1];
        qtdPassos[0] = 0;
        Stack pilhaCaminho = new Stack();


        //Iniciar no ponto central da janela (DEVE SER IMPAR)
        int centro = (linhas/2)+1 -1; //-1 para ajuste ao índice base
0

        double[][] matrizIntensidadeAcumulada = new
double[linhas][colunas];

        if(DOCMtype == 'b')  // [DOCMb]
        {
            // Guarda a mínima intensidade acumulada até o ponto
            // Só tem serventia para o algoritmo de percurso

inicializaMinMatriz(matrizIntensidadeAcumulada,linhas,colunas);
        } else //[DOCMd]
        {
            // Guarda a máxima intensidade acumulada até o ponto

inicializaMaxMatriz(matrizIntensidadeAcumulada,linhas,colunas);
        }

        //Já preenche intensidade Mínina no ponto central
        matrizIntensidadeAcumulada[centro][centro] =
janela[centro][centro];

        // Matriz que guarda os DOC do centro até o ponto
        ArrayList[][] matrizDOC = new ArrayList[linhas][colunas];
        inicializaDOCMatriz(matrizDOC,linhas,colunas);

        // DOCM é obtido pelos máximos da matriz de DOC (em cada
posição)
        double[][] outputDOCM = new double[linhas][colunas];


        // INICIO DO PERCURSO
```

```
        proximoPonto(centro, centro, janela, linhas, colunas, true,
qtdPassos, pilhaCaminho, matrizIntensidadeAcumulada, matrizDOC,
DOCMtype);

        // Já temos os DOCS entre os pontos
        // Queremos o DOCM a partir do MAX
        calculaDOCM(matrizDOC, outputDOCM, linhas, colunas, DOCMtype);

        return outputDOCM;
    }

    // Retorna se está no ponto final ou não
    // Retorno FALSE continua; TRUE PARA
    public static boolean proximoPonto(int x, int y, double[][]
matriz, int qtdLinhas, int qtdColunas, boolean imprimePassos,
            int[] passos, Stack pilhaCaminho, double[][]
matrizIntensidadeAcumulada, ArrayList[][] matrizDOC, char DOCMtype)
    {
        passos[0]++;

        // VISITOU
        pilhaCaminho.push(x*10+y);

        if(imprimePassos){
            imprimeMatriz(matriz, qtdLinhas, qtdColunas);
        }


        //Verificação da Intensidade Mínima  [DOCMb]
        // Esse ponto tem menor intensidade entre todos já
percorridos?
        if(DOCMtype == 'b')
        {
            double minAnterior = minMatriz(matrizIntensidadeAcumulada,
qtdLinhas, qtdColunas);
            double minAtual = minAnterior;
            if(matriz[x][y]<minAnterior)
            {
                matrizIntensidadeAcumulada[x][y] = matriz[x][y];
                minAtual = matriz[x][y];
            }

            // Cálculo deste DOC (Centro até aqui; este ponto) POR
ESTE CAMINHO
            // Será a intensidade mínima acumulada até este ponto;
neste caminho
            matrizDOC[x][y].add(minAtual);
        } else //DOCMd
        {
            double maxAnterior = maxMatriz(matrizIntensidadeAcumulada,
qtdLinhas, qtdColunas);
            double maxAtual = maxAnterior;
            if(matriz[x][y]>maxAnterior)
            {
                matrizIntensidadeAcumulada[x][y] = matriz[x][y];
                maxAtual = matriz[x][y];
            }

            // Cálculo deste DOC (Centro até aqui; este ponto) POR
ESTE CAMINHO
```

```
            // Será a intensidade mínima acumulada até este ponto;
neste caminho
            matrizDOC[x][y].add(maxAtual);
        }

        // Direita
        // Condições: FRONTEIRA(DA JANELA) && PONTO NÃO VISITADO
        if(y+1<=qtdLinhas-1 && (!pilhaCaminho.contains(x*10+y+1)))
        {
            if(proximoPonto(x, y+1, matriz, qtdLinhas, qtdColunas,
imprimePassos, passos, pilhaCaminho, matrizIntensidadeAcumulada,
matrizDOC, DOCMtype))
            {
                return true;
            }
            else
            {
                // Remove o último ponto visitado (para que ele possa
ser visitado futuramente por um caminho diferente deste)
                // Observe que ele não é o ponto atual.
                pilhaCaminho.pop(); // Mas o ponto em [x][y+1]
            }
        }
        // Baixo
        if(x+1<=qtdColunas-1 && (!pilhaCaminho.contains((x+1)*10+y)))
        {
            if(proximoPonto(x+1, y, matriz, qtdLinhas, qtdColunas,
imprimePassos, passos, pilhaCaminho, matrizIntensidadeAcumulada,
matrizDOC, DOCMtype))
            {
                return true;
            }
            else
            {
                pilhaCaminho.pop();
            }
        }
        // Esquerda
        if(y-1>=0 && (!pilhaCaminho.contains(x*10+y-1)))
        {
            if(proximoPonto(x, y-1, matriz, qtdLinhas, qtdColunas,
imprimePassos, passos, pilhaCaminho, matrizIntensidadeAcumulada,
matrizDOC, DOCMtype))
            {
                return true;
            }
            else
            {
                pilhaCaminho.pop();
            }
        }
        // Cima
        if(x-1>=0 && (!pilhaCaminho.contains((x-1)*10+y)))
        {
            if(proximoPonto(x-1, y, matriz, qtdLinhas, qtdColunas,
imprimePassos, passos, pilhaCaminho, matrizIntensidadeAcumulada,
matrizDOC, DOCMtype))
            {
                return true;
            }
```

```
            else
            {
                pilhaCaminho.pop();
            }
        }

        // NÃO TEM MAIS PARA ONDE IR
        if(DOCMtype == 'b')
        {
            matrizIntensidadeAcumulada[x][y] = Double.MAX_VALUE;
//Ponto descartado neste caminho
        } else // [DOCMd]
        {
            matrizIntensidadeAcumulada[x][y] = Double.MIN_VALUE;
//Ponto descartado neste caminho
        }
        return false;
    }

    public static void imprimeMatriz(double[][] matriz, int numLinhas,
int numColunas)
    {
        for(int i = 0; i< numLinhas; i++)
        {
            for(int j = 0; j<numColunas;j++)
            {
                System.out.print(matriz[i][j]);
            }
            System.out.println();
        }
        System.out.println();
        System.out.println();
    }

    public static double minMatriz(double[][] matriz, int numLinhas,
int numColunas)
    {
        double minValue;
        minValue = Double.MAX_VALUE;
        for(int i = 0; i< numLinhas; i++)
        {
            for(int j = 0; j<numColunas;j++)
            {
                if(matriz[i][j]<minValue)
                {
                    minValue = matriz[i][j];
                }
            }
        }

        return minValue;
    }

    public static double maxMatriz(double[][] matriz, int numLinhas,
int numColunas)
    {
        double maxValue;
        maxValue = Double.MIN_VALUE;
        for(int i = 0; i< numLinhas; i++)
        {
```

```
            for(int j = 0; j<numColunas;j++)
            {
                if(matriz[i][j]>maxValue)
                {
                    maxValue = matriz[i][j];
                }
            }
        }

        return maxValue;
    }

    public static void inicializaMinMatriz(double[][] matriz, int
numLinhas, int numColunas)
    {
        for(int i = 0; i< numLinhas; i++)
        {
            for(int j = 0; j<numColunas;j++)
            {
                matriz[i][j] = Double.MAX_VALUE;
            }
        }
    }

    public static void inicializaMaxMatriz(double[][] matriz, int
numLinhas, int numColunas)
    {
        for(int i = 0; i< numLinhas; i++)
        {
            for(int j = 0; j<numColunas;j++)
            {
                matriz[i][j] = Double.MIN_VALUE;
            }
        }
    }

    public static void inicializaDOCMatriz(ArrayList[][] matriz, int
numLinhas, int numColunas)
    {
        for(int i = 0; i< numLinhas; i++)
        {
            for(int j = 0; j<numColunas;j++)
            {
                matriz[i][j] = new ArrayList();
            }
        }
    }

    public static void imprimePilha(Stack pilha)
    {
        while(!pilha.empty())
        {
            System.out.println(pilha.pop());
        }
    }

    public static void calculaDOCM(ArrayList[][] matrizDOC, double[][]
outputDOCM, int numLinhas, int numColunas, char DOCMtype)
    {
        for(int i = 0; i< numLinhas; i++)
```

```
        {
            for(int j = 0; j<numColunas;j++)
            {
                if(DOCMtype == 'b')
                {
                    outputDOCM[i][j] =
(double)Collections.max(matrizDOC[i][j]);
                } else // [DOCMd]
                {
                    outputDOCM[i][j] =
(double)Collections.min(matrizDOC[i][j]);
                }
            }
        }
    }


    public static void preencheMatriz(double[][] matriz, int
numLinhas, int numColunas)
    {
        int cont = 0;
        for(int i = 0; i< numLinhas; i++)
        {
            for(int j = 0; j<numColunas;j++)
            {
                matriz[i][j] = cont;
                cont++;
            }
        }
    }

}
```