UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALÉCIO SANTOS BARROS

UTILIZAÇÃO DA MORFOLOGIA FUZZY PARA QUANTIFICAÇÃO DE ESPOROS DE FUNGOS MICORRIZICOS

VITORIA DA CONQUISTA-BA
2014

ALÉCIO SANTOS BARROS

UTILIZAÇÃO DA MORFOLOGIA FUZZY PARA QUANTIFICAÇÃO DE ESPOROS DE FUNGOS MICORRIZICOS

Monografia apresentada ao Curso de Graduação em Ciência da Computação da Universidade Estadual do Sudoeste da Bahia, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.
Orientador(a): Prof. Dr. Roque Mendes Prado Trindade

VITORIA DA CONQUISTA-BA
2014

Resumo: Este trabalho apresenta um método de contagem de esporos de fungos micorrizicos a partir de imagens digitais utilizando a morfologia *Fuzzy* para separação dos esporos e algoritmos desenvolvidos em Scilab para a quantificação. O objetivo é mostrar a eficácia da morfologia *Fuzzy* e das funções desenvolvidas na separação e contagem. A estratégia metodológica desenvolvida foi a de utilizar as definições de morfologia *Fuzzy* desenvolvidas por ANDRADE (2014) em seu trabalho *Um Sistema de Contagem baseado em Morfologia Matemática Fuzzy* para desenvolver um algoritmo capaz de separar e contar fungos. Espera-se com isso obter um método capaz de realizar uma contagem satisfatória.

Abstract: This work presents a method of fungal spore counts mycorrhizal from digital images using the *Fuzzy* morphology for separation of spores and algorithms developed in Scilab for quantification. The goal is to show the effectiveness of *Fuzzy* morphology and functions developed for the separation and counting. he developed methodological strategy was to use the *Fuzzy* morphology settings set by ANDRADE (2014) in his work A System Count Based on *Fuzzy* Mathematical Morphology to develop an algorithm able to separate and count fungi. It is hoped that a method capable of obtaining a satisfactory hold count.

Sumário

Introdução 1

A área de processamento de imagens digital tem um papel importante na vida humana. Atualmente, o campo de processamento de imagens tem inúmeras aplicações comerciais, científicas, industriais, médicas, militares, de entretenimento e outras. As pesquisas científicas, e o desenvolvimento de novas tecnologias de alto padrão resultaram em tais aplicações. Essa interação continua levou a uma área de pesquisa ativa e bastante ampla. Alguns tópicos bem conhecidos em processamento de imagens digitais são melhoria de qualidade da imagem (filtragem, redução de ruídos, realce, restauração), analise de imagem (detecção de borda, segmentação, reconhecimento e contagem de objetos, interpretação), compressão de imagem e reconstrução de imagem.

A fim de lidar com todos esses e outros problemas, varias técnicas foram desenvolvidas e introduzidas, muitas vezes com grande sucesso. Dentre as diferentes técnicas que estão atualmente em uso, encontram-se as técnicas *fuzzy*. Bouchet et al (2007) propuseram um método para segmentação de imagens de angiografia retinal usando morfologia *fuzzy* para melhorar o diagnóstico de profissionais da saúde. Boaventura (2010) propõe uma técnica pra processamento de imagem e detecção de borda utilizando números *fuzzy* e morfologia matemática *fuzzy*. Estes são alguns exemplos do uso de tal técnica. Baseado em muitas experiências bem sucedidas em aplicações da teoria de conjunto *fuzzy*, as técnicas *fuzzy* agregam efetivamente valores as áreas de processamento de imagem e visão computacional.

Uma das principais aplicações do processamento digital de imagens é a contagem de certos objetos em uma imagem. O termo “objeto” refere-se aqui a qualquer elemento de interesse que pode ser identificado numa imagem digital. A variedade de problemas de contagem de objetos é muito grande, cada qual com suas características e desafios intrínsecos. O grau de dificuldade do problema depende de uma série de fatores: contraste entre os objetos e o fundo, grau de agrupamento dos objetos, textura dos objetos e sua variação, tamanho do objeto e sua variação, complexidade dos objetos, etc.

Varias estratégias foram propostas para solucionar uma grande variedade de problemas de contagem de objetos como células, bactérias, árvores, frutas, amostra de solo, fungos, pólen, espigas, entre outras (BARBEDO 2012). Salis & Pereira (2006) propuseram um método para a contagem automática de tarugos de aço na linha de produção de

siderúrgicas, obtendo bons resultados utilizando ferramentas como filtragens para a remoção de ruído e correção de brilho, limiarização, erosão e dilatação. Poomcokrak & Neatpisarnvanit (2008) desenvolveram um método simples de contagem de células vermelhas do sangue que consiste em eliminar as células incompletas, extrair as células individuais com detecção de borda usando morfologia matemática e identificação e contagem usando redes neurais. Men et al (2008) apresentaram um método para contagem de bactérias heterotróficas em água de refrigeração industrial que utiliza a conversão para tons de cinza e binarização usando o método de limiar com ponderação pela escala de cinza. Os trabalhos citados são alguns dentre muitos que se utilizam de técnicas de processamento de imagem para contagem de objetos.

Este trabalho tem como objetivo mostrar a eficiência dos operadores morfológicos *fuzzy* propostos por ANDRADE (2014), na separação de objetos, para isso serão realizados testes buscando identificar o melhor operador e o melhor elemento. Um algoritmo de contagem será utilizado como meio de medir esta eficiência.

1.2 Organização do trabalho

Este trabalho está organizado da seguinte forma: Capítulo 2 (Referencial teórico) foi dividido em sessões que abordam os tópicos referentes a lógica *fuzzy*, conjunto *fuzzy*, imagens digitais, vizinhança de imagens binárias, morfologia, operadores morfológicos binários de erosão e dilatação, operadores morfológicos em escala de cinza de erosão e dilatação e operadores morfológicos *fuzzy* de erosão e dilatação, a ferramenta scilab e os fungos micorrízicos. Capitulo 3 (Materiais e métodos) abordará o tipo de pesquisa, as ferramentas e material utilizados para desenvolver a mesma. Capitulo 4 (Desenvolvimento) ira apresentar o desenvolvimento dos algoritmos e os testes dos mesmos. Capitulo 5 (Resultados e discussões) apresenta e discuti os resultados das técnicas abordadas. Por fim, a conclusão e os trabalhos futuros serão apresentados no Capítulo 6.

2 Referencial teórico

2.1 Logica *Fuzzy*

A lógica *fuzzy* é a lógica baseada na teoria dos conjuntos *fuzzy*. Essa foi criada por volta de 1920 quando um polonês chamado Jan Luasiewicz (1878-1956) utilizando-se do principio da incerteza apresentou pela primeira vez as noções da lógica dos conceitos vagos na qual é admissível um conjunto com valores não precisos, porem, só a partir de 1965 esses conceitos passaram a ser conhecidos quando o professor Lofti Zadeh publicou o artigo *Fuzzy Sets* no jornal *Information and Control* (AGUADO e CANTANHEDE, 2010, ORTEGA, 2001).

O termo *fuzzy* em língua inglesa pode ter vários significados, que variam de acordo com o contexto de interesse, mas o conceito básico deste adjetivo passa sempre pelo domínio semântico do vago, indistinto e incerto. As tentativas de tradução para o português ainda não são unanimidade e parecem tender para expressões do tipo: “nebuloso” e “difuso” que são exemplos mais populares de traduções para *fuzzy*.

Vale ressaltar que para Cox (1994) a principal diferença entre a lógica *fuzzy* e a lógica *booleana* é que está trabalha somente com respostas exatas distanciando-se da realidade enquanto aquela trabalha com respostas relativas tendo a capacidade de se aproximar do mundo real onde não existem somente respostas exatas. Assim, a lógica *fuzzy* dá espaço ao meio termo apresentando ainda a possibilidade de mensurar o grau de aproximação da solução exata e assim inferir algo que seja necessário. No entanto, KLIR (1995) apresenta que o que diferencia a lógica clássica da lógica *fuzzy* está no conjunto que cada uma delas tem como valores verdadeiros ou valores respostas. Enquanto a lógica clássica propõe que esses valores seja verdadeiro ou falso, a lógica *fuzzy* propõe que isso seja uma questão de grau.

Devido à proximidade com os problemas do mundo real e sua adaptabilidade no que concernem as respostas aos problemas, à lógica *fuzzy* vem sendo cada vez mais utilizada em pesquisas e se reformulando ao longo do tempo. O ápice de seu crescimento deu-se durante a década de 80, principalmente, no Japão onde foi criado o primeiro grupo de pesquisas em sistemas *fuzzy*, coordenado pelo professor Toshiro Terano (ORTEGA, 2001). Outra característica dessa lógica é sua versatilidade o que pode ser visualizado nas diversas áreas onde se encontram aplicações da lógica *fuzzy* como: engenharia, química, biologia, medicina, epidemiologia, ecologia, economia, psicologia, ciências sociais entre outros, ganhando maior

espaço atualmente em otimizações e automação industrial devido sua facilidade de retratar a lógica da racionalidade humana ao resolver problemas (ORTEGA, 2001).

2.1.1 Conjunto *fuzzy*

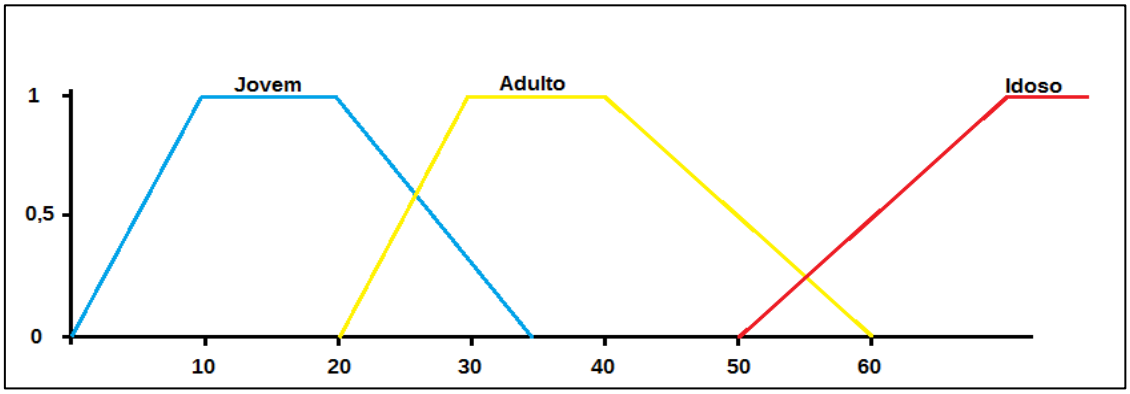
Definiremos, agora, alguns elementos constituintes da lógica, aqui, utilizada, demonstrando suas unidades e aplicabilidade. Na teoria de conjuntos clássica, um elemento pertence ou não a um dado conjunto. Dado um universo U e um elemento particular x ∈ U, o grau de pertinência μA(x) com respeito a um conjunto A ∈ U é dado por:

$$\mu A(x) = \{1 \text{ se } x \in A, 0 \text{ se } x \notin A\}$$

A função μA(x): U → {0,1} é chamada de função característica na teoria clássica de conjuntos (ORTEGA, 2001). Uma generalização desta ideia é frequentemente utilizada para manipulação de dados com erros limitados na qual todo número dentro de um erro percentual tera um fato de pertinência 1, tendo todos os demais um fator de pertinência 0. Para o caso preciso, o fator de pertinência é 1, somente no número exato, sendo 0 para todos os demais.

Para fazer uma caracterização menos restrita, Zadeh (ZADEH, 1965) sugere que alguns elementos pertençam mais a um conjunto do que a outros. O fator de pertinência pode então assumir qualquer valor entre 0 e 1, sendo que o valor 0 indica uma completa exclusão e um valor 1 representa completa pertinência. Esta generalização aumenta o poder de expressão da função característica. Para exemplificar as funções de pertinência, considere a **Figura 1**, onde através de alguns gráficos são representadas três variáveis etárias: jovem, adulto e idoso. Estas variáveis são relativas a uma análise sobre a faixa etária onde o eixo das ordenadas representa a idade e o das abscissas o grau de pertinência.

Figura 1: Gráfico de pertinência etária



No exemplo acima uma pessoa com 51 anos pode ser representada {0, 0.45, 0.03} em relação a pertinência ao conjunto *fuzzy* de faixa etária.

2.2 Imagens digitais

Este tópico trás as definições básicas de uma imagem digital e as possíveis definições para a vizinhança de *pixel*. Uma imagem digital é uma função bidimensional da intensidade da luz f(x,y), onde x e y denotam as coordenadas espaciais (largura e altura) e o valor f em qualquer ponto (x, y) é proporcional ao brilho (ou nível de cinza) da imagem naquele ponto (GONZALES, 2002).

As imagens digitais são compostas por elementos chamados *pixels* e seus valores são definidos de formas diferentes a depender do tipo de imagem. Nas imagens binárias os valores são limitados a 0 e 1 (branco e preto). Nas imagens em escala de cinza, os valores dos *pixels* representam seus níveis de cinza, tipicamente as imagens em escala de cinza são representados por 8 *bits* por *pixel*, o que permite 256 níveis (GONZALES, 2002). Nas imagens coloridas, os valores dos *pixels* são definidos como t-uplas ternárias, sendo que os seus valores variam com o espaço de cores utilizado.

2.2.1 Vizinhança imagens binárias

A fim de identificar objetos em um padrão digital, precisamos localizar grupos de *pixels* pretos (valor 1) que estão "conectados" uns aos outros. Em outras palavras, os objetos em um determinado padrão digital são os componentes conectados em um mesmo padrão.

Em geral, um componente conectado é um conjunto de *pixels* pretos, P, tal que, para cada par de *pixels* pi e pj em P, existe uma sequencia de *pixels* pi, ..., pj tal que:

a) todos os *pixels* da sequencia estão no conjunto P, ou seja, são pretos, e

b) a cada 2 *pixels* que são adjacentes na sequencia são "vizinhos"

Como resultado, uma importante questão que surge é: Quando é que podemos dizer que dois *pixels* são "vizinhos"?

Como estamos usando *pixels* quadrados, a resposta à pergunta anterior não é trivial. A razão para isso é a seguinte: em uma matriz de imagem, os *pixels* ou compartilham uma aresta, um vértice, ou nenhum. Há 8 *pixels* que compartilham uma aresta ou um vértice com

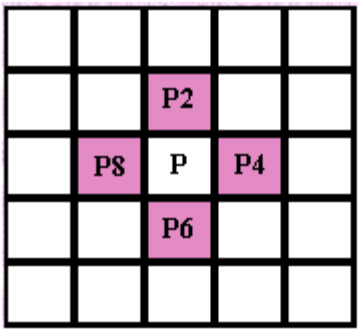
um *pixel*; esses *pixels* compõem vizinhança. A partir destes 8 *pixels* duas definições de vizinhança foram construídas $N_4(p)$ e $N_8(p)$ (GONZALES, 2002).

(a) $N_4(p)$. Defini-se que um *pixel* q é vizinho de um *pixel* j caso q e j compartilhem uma aresta.

(b) $N_8(p)$. Defini-se que um *pixel* q é vizinho de um *pixel* j caso q e j compartilhem uma aresta ou um vértice.

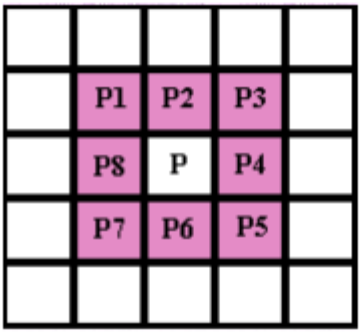
As figuras, **Figura 2 e Figura 3** abaixo mostram a área de vizinhança para as definições citadas.

Figura 2: P1, P2, P3, P4 são pixel vizinhos a P.



$N_4(p)$

Figura 3: P1, P2, P3, P4, P5, P6, P7, P8 são pixel vizinhos a P.



$N_8(p)$.

2.3 Morfologia

Agora, voltaremos nossa atenção para outro ponto concernente a nossa investigação, o estudo da forma dos elementos, ou seja, a morfologia. A morfologia matemática (doravente MM) surgiu na década de 1960 e sua teoria foi formulada por Georges Matheron e Jean Serra, entretanto, somente o advento dos computadores pôde torná-la prática. Inicialmente, ela foi desenvolvida para manipular imagens binárias, e, posteriormente, estendida para imagens em tons de cinza e colorida (MATTA, 1998). As teorias para esta extensão se baseiam nas seguintes abordagens (ANDRADE, 2014):

1. Umbra. Uma imagem em escala de cinza é considerada como uma paisagem em três dimensões utilizando ferramentas que são conhecidas como bidimensional.

2. Conjuntos Limiar. Em cada conjunto, no qual a imagem foi decomposta, pode se aplicar um operador binário, que sintetiza uma transformada em escala de cinza da imagem.

3. Reticulados Completos. Abordagem algébrica de morfologia matemática utiliza reticulados completos como ferramentas matemáticas para a produção de operadores morfológicos.

4. Lógica *Fuzzy*. Utiliza a lógica *fuzzy* e conjuntos *fuzzy* para a construção dos operadores.

A ideia por trás da morfologia matemática é considerar que as imagens são constituídas por um conjunto de elementos, os *pixels*, que se agrupam numa estrutura bidimensional (RIBEIRO, 2001). Os mais antigos usos da palavra morfologia estão relacionados com biologia, geologia e linguística. Em biologia, morfologia está relacionada ao estudo da forma dos seres vivos, ou de parte dele. A forma de uma folha pode ser usada para identificar uma planta. Na geologia, a morfologia trata de estudar a origem e a evolução da superfície terrestre. Por exemplo, graças a ela podemos conhecer como se originou as diversas cadeias montanhosas. Em linguística a morfologia é o ramo que trata do estudo da estrutura interna das palavras, para desta maneira, definir, delimitar e classificar as unidades que a compõem, ou seja, em termos gerais, é simplesmente o estudo da palavra (MORFOLOGIA, s.d).

A MM tem como objetivo extrair informações relativas à geometria e à topologia de um conjunto desconhecido de uma imagem. A grande chave da morfologia matemática é o elemento estruturante. O elemento estruturante é um conjunto cuja forma e tamanho são conhecidos. Esse elemento é comparado, a partir de uma transformação, ao conjunto desconhecido da imagem (RIBEIRO, 2001). Seu formato e seu tamanho possibilitam testar e quantificar de que maneira o elemento estruturante "está ou não está contido" na imagem. Marcando-se, na imagem resultante, as posições onde o elemento estruturante inclui-se na imagem original, obtém-se uma primeira resposta sobre a estrutura geométrica das entidades presentes na imagem.

2.3.1 Operadores morfológicos

Os operadores morfológicos são funções que atuam sobre uma imagem através de uma pequena imagem que funciona como molde. Erosão e dilatação são os operadores básicos que constituem os pilares da morfologia matemática, a partir destes, outros operadores podem ser criados. Abertura e fechamento são alguns deles.

2.3.1.1 Operadores morfológicos binários

Discutiremos, aqui, os operadores morfológicos binários que foram utilizados em nossa pesquisa e que compõem a metodologia *Fuzzy*.

2.3.1.1.1 Erosão binária

A erosão de um conjunto A (imagem) pelo conjunto B (elemento estruturante) é definida da seguinte forma:

$$ero\ (A) = A\ ero\ B = \{X\ |(B_x) \subseteq A\}$$

Deve-se deslizar o elemento estruturante B sobre a imagem A e para cada *pixel* x que corresponde ao ponto central do elemento estruturante será ativado (valor 1 *pixel* branco) se o elemento estruturante estiver inteiramente contido na imagem original, caso contrário, será marcado como irrelevante e será apagado (valor 0 *pixel* preto) (ANDRADE 2012, RIBEIRO, 2001, MATTA, 1998).

A erosão binária faz desaparecer todos os conjuntos inferiores ao elemento estruturante, e aumenta os "furos" interiores aos conjuntos. Permite também separar conjuntos que estiverem próximo. Desta forma tem como resultado uma imagem menor ou igual a imagem original, em todos os casos (MATTA, 1998).

2.3.1.1.2 Dilatação

A dilatação de um conjunto A (imagem) pelo conjunto B (elemento estruturante) é definida da seguinte forma:

$$dil(A) = A\ dil\ B = \{X\ |(B_x^{\wedge}) \cap A \neq \emptyset\}$$

Da mesma forma como na erosão, o elemento estruturante desliza sobre a imagem. Se houver alguma interseção do elemento estruturante com a imagem, o *pixel* x da imagem correspondente ao ponto central do elemento estruturante será ativado, caso contrário será marcado como irrelevante (ANDRADE 2012, RIBEIRO, 2001, MATTA, 1998).

A imagem dilatada tem resultado contrario a erosão tendo como resultado uma imagem sempre maior ou igual à imagem original. Permite conectar conjuntos separados e preenche todos os furos que são menores do que o elemento estruturante (MATTA, 1998).

2.3.1.2 Operadores morfológicos em níveis de cinza

Imagens digitais em nível de cinza podem ser representadas por conjuntos cujos componentes estejam em Z^3. O gráfico de uma imagem I pode ser representado da seguinte forma $G(I) = \{x, g(x) | x \in Z^2\}$ onde *x* representa as coordenadas e $g(x) \in \{0,1,2, \dots ,255\}$ onde $g: G \subset Z^2 \rightarrow [N_{min}, N_{max}]$, a função que denota a intensidade de cinza (ANDRADE 2014).

2.3.1.2.1 Erosão

Sendo I(x, y) a imagem e B(x, y) o elemento estruturante a erosão é definhada por:

$$(I \; ero \; B) = \min \{I(s - x , t - y) - B(x , y) \mid (s - x , t - y) \in I , (x, y) \in B\}$$

Wangenheim (WANGENHEIM, s.d) define os seguintes passos para realizar a operação de erosão:

1. Posiciona-se a origem do elemento estrutural sobre o primeiro *pixel* da imagem que sofre erosão.
2. Calcula-se a diferença de cada par correspondente de valores de *pixels* do elemento estrutural e da imagem.
3. Acha-se o valor mínimo de todas essas diferenças, e armazena-se o *pixel* correspondente na imagem de saída para este valor.
4. Repete-se este processo para cada *pixel* da imagem que sofre erosão

2.3.1.2.2 Dilatação

Sendo I(x, y) a imagem e B(x, y) o elemento estruturante a dilatação é definhada por:

$$(I \; dil \; B) = \max \{I(s - x, t - y) + B(x, y) \mid (s - x , t - y) \in I , (x, y) \in B\}$$

Wangenheim (WANGENHEIM, s.d) define os seguintes passos para realizar a operação de dilatação:

> 1. Posiciona-se a origem do elemento estrutural sobre o primeiro *pixel* da imagem a ser dilatada.
> 2. Calcula-se a soma de cada par correspondente de valores de *pixels* do elemento estrutural e da imagem.
> 3. Acha-se o valor máximo de todas essas somas, e armazena-se o *pixel* correspondente na imagem de saída para este valor.
> 4. Repete-se este processo para cada *pixel* da imagem a ser dilatada.

2.3.1.3 Operadores morfológicos Fuzzy

Diferente das imagens binárias onde o valor do *pixel* pode assumir apenas dois valores as imagens em tons de cinza compreendem um maior intervalo indo de 0 a 255, o que impossibilita a aplicação da lógica booleana. Neste caso, a lógica *fuzzy* pode ser utilizada necessitando apenas que a imagem seja normalizada para que os valores de *pixel* fiquem dentro do intervalo [0,1]. Assim, uma função de normalização deve ser definida onde $v:\{1,2,3,...,255\}\rightarrow[0,1]$ (ANDRADE 2012).

A seguir são apresentadas as definições das funções usadas no desenvolvimento deste trabalho.

$$f(x) = sen(\frac{x}{255} * \frac{\pi}{2}) \qquad \text{I}$$

$$f(x) = \frac{x}{255} \qquad \text{II}$$

$$f(x) = \frac{1}{2}\left\{\tan\left[\left(\frac{1}{360}\left(\frac{x}{255} * 90 + 135\right) * 2\pi\right) + 1\right]\right\} \qquad \text{III}$$

Tais funções serão chamadas mais a frente como seno(x), x/255 e tangente(x) respectivamente.

2.3.1.3.1 Erosão e Dilatação *fuzzy*

Sem entrar em detalhes sobre os fundamentos teóricos, a seguir definem-se duas operações morfológicas básicas.

A erosão *fuzzy* de uma imagem g pelo elemento estruturante B no ponto x é definida como

$$\varepsilon^F(g,B)(x) = \underset{y \in g}{inf}\{I(B_x(y), g(y))\}$$

onde I denota uma *Implicação Binária* (BOUCHET 2007).

A dilatação *fuzzy* de uma imagem g pelo elemento estruturante B no ponto x é definida como

$$\delta^F(g,B)(x) = \underset{y \in g}{sup}\{C(B_x(y), g(y))\}$$

onde C denota uma *Conjunção Binária* (BOUCHET 2007).

Um aprofundamento no que concerne a definição dos operadores morfológicos da erosão e da dilatação *fuzzy* usando as implicações e conjunções *fuzzy* respectivamente, foi elaborado por ANDRADE (2014) e suas definições serão mostradas abaixo.

(a) A erosão Gödel de uma imagem g pelo elemento estruturante B e dada por:

$$\varepsilon^{GD} g(x) = \underset{y \in g}{inf} \begin{cases} 1 \quad se \quad Bx(y) \leq g(y) \\ g(y) se \quad Bx(y) < g(y) \end{cases}$$

(b) A dilatação Gödel de uma imagem g pelo elemento estruturante B é dada por:

$$\delta^{GD} g(x) = \underset{y \in g}{sup}[\min[Bx(y), gx(y)]]$$

(c) A erosão Goguen de uma imagem g pelo elemento estruturante B e dada por:

$$\varepsilon^{GG} g(x) = \underset{y \in g}{inf} \begin{cases} 1 \quad se \quad B(x) \leq g(y) \\ \dfrac{g(y)}{B(y)} \; se \quad B(x) < g(y) \end{cases}$$

(d) A dilatação Goguen de uma imagem g pelo elemento estruturante B e dada por:

$$\delta^{GG} g(x) = \underset{y \in g}{sup}\,[B(y) \cdot g(y)]$$

(e) A erosão Lukasiewisz de uma imagem g pelo elemento estruturante B e dada por:

$$\varepsilon^{LK} g(x) = \underset{y \in g}{inf}\,[1, 1 - B(y) + g(y)]$$

(f) A dilatação Lukasiewisz de uma imagem g pelo elemento estruturante B e dada por:

$$\delta^{LK} g(x) = \underset{y \in g}{sup} [0, B(y) + g(y) - 1]$$

2.4 Scilab

Nessa sessão trataremos a respeito da ferramenta Scilab, que é um software livre e de código aberto para computação numérica proporcionando um ambiente de computação poderosa para aplicações de engenharia e ciência. Essa ferramenta possui uma linguagem de programação de alto nível, orientada à análise numérica. Essa linguagem provê um ambiente para interpretação, com diversas ferramentas numéricas como, por exemplo: algoritmos complexos podem ser criados em poucas linhas de código, em comparação com outras linguagens como C, Fortran, ou C++. Desenvolvido desde 1990 pelos pesquisadores do INRIA (Institut National de Recherche en Informatique et en Automatique) e do ENPC (École Nationale des Ponts et Chaussées), então pelo Consorcio Scilab desde Maio de 2003, Scilab é agora mantido e desenvolvido pelo Scilab Enterprises desde Julho de 2012.

Distribuido gratuitamente via Internet desde 1994, o Scilab é atualmente usado em diversos ambientes industriais e educacionais pelo mundo. Além da distribuição com o código fonte, existem, também, distribuições pré-compiladas do Scilab para vários sistemas operacionais (FILHO, s.d).

2.5 Fungos

Outro ponto que deve ser definido em nosso trabalho é referente ao nosso corpora de análise que é constituído pela contagem de fungos *Micorrizicos Arbusculares*. Nesse ponto, passaremos a definir o que são fungos e como é feita a contagem que aqui será efetuada. A maioria dos organismos vivos estabelecem associações com outros organismos com a finalidade de garantir a sobrevivência e entre estas estão às associações micorrízicas do grego mico [fungo] e riza [raiz]. Van der Heijden (JAKOBSEN, 2002) vai além das relações funcionais que se estabelecem entre fungos e plantas, "associações micorrízicas devem sempre ser consideradas quando se busca entender a ecologia e evolução de plantas, suas

comunidades e ecossistemas". Há experimentos que mostram o papel dessa simbiose no resultado da competição e sucessão de plantas, bem como na hipótese de que a evolução de plantas terrestres tem sido dependente da presença dessa simbiose (JAKOBSEN, 2002).

Vários são os tipos de associações micorrízicas, sendo algumas delas muito específicas, encontradas em apenas algumas famílias de plantas terrestres (BERBARA, 1996), no entanto levantamentos indicam que 80 % das famílias de plantas são formadas por espécies que formam micorrizas arbusculares (doravante MA) que são encontradas em todas as latitudes e em quase todos os ecossistemas terrestres (BERBARA, 1996). Os fungos MAs são classificados como pertencentes ao filo Glomeromycota, classe Glomeromycetes e agrupados em quatro ordens, 11 famílias e 25 gêneros (REDECKER, 2013) e mais de 200 espécies.

Os esporos mais especificamente chamados de glomerosporos (GOTO, 2006) germinam e suas hifas se desenvolvem junto ao sistema radicular das plantas. Nessa associação, a planta se beneficia pelo aumento de absorção de água e nutrientes, principalmente de fósforo (P), pelas hifas fúngicas, enquanto o fungo obtém da planta os fotoassimilados necessários para que se complete seu ciclo de vida (SIQUEIRA, 2006). Em áreas de solo de baixa fertilidade natural, as deficiências nutricionais são umas das principais limitações ao crescimento de espécies arbóreas nativas. O fósforo é o nutriente que requer maior atenção devido ao baixo teor nos solos das regiões tropicais (BRITO, 2013). Vários estudos relacionam a nutrição de diversas essências florestais e frutíferas à presença de simbioses mutualística, como as MAs, que são capazes de diminuir as deficiências nutricionais que ocorrem nos solos de baixa fertilidade (SIQUEIRA, 2002). Esta melhor nutrição está associada ao melhor incremento na absorção de nutrientes, principalmente fósforo (SOUZA, 2008). Em vista dos benefícios propiciados aos hospedeiros, os FMA são de grande interesse para as regiões tropicais, especialmente para o Brasil, devido às condições ambientais e à baixa fertilidade de muitos solos (SIQUEIRA, 2002).

Um dos procedimentos metodológicos básicos ao estudo das micorrizas arbusculares é a extração dos glomerosporos do solo (peneiramento úmido), sendo proposta por Gerdeman e Nicolson (GERDEMAN, 1963), com a contribuição de Jenkins (JENKINS, 1964), mais aceita e difundida mundialmente. O método do peneiramento úmido consiste em pegar porções de terra com massa conhecida mistura-las em água sendo a mistura passadas através de peneiras e centrifugadas repetidas vezes posteriormente uma solução de sacarose é acrescentado à mistura para as ultimas centrifugações. Ao final deste procedimento, será obtida uma determinada quantidade de glomerosporos que necessitara ser quantificada. Essa contagem é realizada manualmente. Os glomerosporos são transferidos para uma placa de Petri e para

auxiliar na visualização dessas estruturas que variam de 22 a 1050μm (SOUZA, 2010) um microscópio estereoscópico (lupa) é utilizado. Outros tipos de auxílios também são utilizados como o uso de uma folha quadriculada sob a placa de Petri para servir de orientação ou placa de Petri com círculos concêntricos (placa canelada). As canaletas formadas pelos círculos em alto relevo ajudam na separação de glomerosporos facilitando sua contagem. Embora haja estes auxílios é comum que haja erros e frequentemente uma segunda contagem na mesma amostra resulta em valor diferente. Levando-se em consideração o numero de amostras que normalmente são analisadas a contagem destas estruturas diminutas trata-se de um trabalho cansativo e desgastante.

3 Material e métodos

3.1 Metodologia

No presente trabalho foi utilizada a pesquisa bibliográfica, que constitui parte da pesquisa exploratória. A pesquisa bibliográfica é parte obrigatória por ser ela de natureza teórica, e é por meio da bibliografia consultada que se toma conhecimento sobre a produção científica que já existe.

Para Gil (2004) as pesquisas classificam-se em três grandes grupos: exploratórias, descritivas e explicativas. A pesquisa atual se classifica como exploratória, pois de acordo com o mesmo autor, é uma das pesquisas que possibilita ao pesquisador a criar, esclarecer e modificar conceitos e ideias com o objetivo de ampliar e/ou formular novos problemas ou hipóteses que possibilite ser estudada posteriormente em outras pesquisas mais consistentes.

Gil (2004) acrescenta ainda, que a pesquisa exploratória objetiva proporcionar maior familiaridade com o problema, com vistas a torná-lo explícito ou a construir hipóteses, tendo como objetivo principal o aprimoramento de ideias ou a descoberta de intuições.

3.3 Preparação do ambiente.

Para desenvolvimento deste trabalho foi utilizado o software Scilab e com este foram implementados os códigos dos operadores morfológicos de erosão e dilatação, e das rotinas para o processamento das imagens e contagem dos esporos de fungos.

A ferramenta Scilab utilizada foi o Scilab 5.4.1 versão 64 *bits* pré-compilada baixada a partir do site www.scilab.org. Também foi utilizado um modulo adicional o SIVP - Scilab Image and Video Processing Toolbox este modulo possui funções para o processamento de imagem e foi utilizado para facilitar a leitura e gravação das imagens. Este módulo pode ser instalado a partir do próprio Scilab com seu gerenciador de módulos ATOMS.

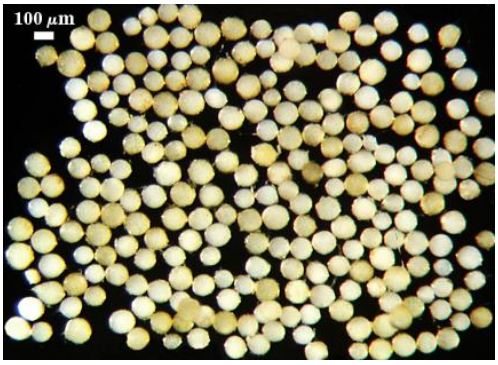
3.2 As imagens

Para desenvolvimento do trabalho foram utilizadas imagens do IVAM (International Culture Collection of Arbuscular Mycorrhizal Fungi) pertencente a Universidade Oeste da Virginia (WVU). As imagens foram retiradas do Site: *invam.wvu.edu.*

As imagens dos esporos apresentam algumas características similares. Todas as imagens têm aproximadamente 400 x 300 *pixel* de tamanho e um fundo escuro tendendo ao preto. Os esporos estão dispostos na imagem de forma que muitos estão próximos entre si chegando a se tocarem e em alguns casos sobrepondo parte um do outro. A cor dos esporos e o tamanho não apresentam características muito similares, dentro do conjunto de imagens são encontradas varias cores e tamanhos. As figuras, **Figura 4 e Figura 5** abaixo mostram estas características.

Figura 4: imagem (430x311) esporos fungo claroidwh.

Figura 5: imagem (387x303) esporos fungo laeviswh.



4 Processo de quantificação de esporos

4.1 Desenvolvimento dos algoritmos

A maioria dos códigos desenvolvidos tomou como base as definições dos operadores *fuzzy* e das funções de normalização. Outros algoritmos foram desenvolvidos como a máscara de corte, conversão da imagem para binário, função de contagem e funções que auxiliaram o processo como um todo. A seguir descreveremos alguns dos algoritmos.

4.1.1 Funções de normalização

Abaixo são mostrados trechos dos algoritmos de normalização e de *"desnormalização"* **Trecho código 1** que recebem como parâmetros a matriz contendo a imagem e o número correspondente ao tipo de função usada. Com as facilidades do Scilab não é necessário fazer um laço de repetição para percorrer todos os *pixels* da imagem, pois a função pode ser aplicada diretamente na matriz como pode ser observado na linha 5 e 21 dos trechos de códigos. A Função de normalização converte os valores dos *pixels* que estão no intervalo [0, 255] para o intervalo real [0, 1] possibilitando que a imagem seja processada por operadores morfológicos e a função de *"desnormalização"* retorna os valores para o intervalo [0, 255] para que a imagem possa ser remontada.

Trecho código 1. Algoritmo função de normalização e *desnormalização*.

```
1   // Função que retorna os valores dos pixels (0 a 255) em uma escala de 0 a 1
2   function [matrizNormalizada]=fuzzificacao(matriz, tipoNormalizacao)
3
4   if tipoNormalizacao == 1 //Se a funçao x/255 for escolhida
5       matrizNormalizada = double(matriz./255);
6   end
...
15  endfunction

17   // Função que retorna os valores dos pixels (0 a 1) em uma escala de 0 a 255
18   function [matrizDesnormalizada]=desfuzzificacao(matriz, tipoFuzzificacao)
19
20   if tipoFuzzificacao == 1
21       matrizDesnormalizada = uint8(matriz*255);
22   end
...

30  endfunction
...
```

4.1.2 Operadores

Os algoritmos dos operadores seguem uma estrutura básica igual para todos mudando apenas o seu núcleo onde se calcula o valor do *pixel* o que em cada caso é definido para cada implicação de Goguen, Gödel e Lukasiewicz.

Trecho código 2. Algoritmo que percorre a imagem e o elemento.

```
    for m = 1: linhasMatriz //laços para percorrer todos os pixels da imagem
        for n = 1 : colunasMatriz

            //Para cada pixel da imagem, esses laços percorrem o conjunto dos pixels que
interferem no valor do pixel em questão
            for l = 1: linhasElemento
               for c = 1: colunasElemento
                  ... //instrução que calcula o valor do pixel a partir da definição do operador
               end
            end
             matrizResultante(m,n) = valorFinal; // O pixel recebe o ínfimo dos valores
calculados anteriormente
        end
    end
```

O **Trecho código 2** acima mostra a estrutura básica do código dos operadores, este trecho contem dois laços ‘for’ necessários para percorrer os *pixels* da imagem e outros dois laços ‘for’ para percorrer os *pixels* do elemento estruturante. Dentro do laço mais interno o valor do pixel é calculado e posteriormente o valor é armazenado na matriz resultante.

O algoritmo inicialmente tem ordem de complexidade n^4 considerando que a imagem e o elemento tem tamanho desconhecido, no entanto, para contagem de esporos de fungos o elemento estruturante vai ser definido de forma que seu tamanho e formato serão conhecidos o que leva a ordem de complexidade do algoritmo a ser reduzida para n^2. Os algoritmos que calculam o valor do pixel foram codificados a partir das definições os operados e suas implicações.

Trecho código 3. Algoritmos que implementam as definições dos operadores.

```
    // Definição erosão Godel
     if fuzzyElemento(l,c) <= fuzzyImagem(l,c) // Se elemento(x)<= imagem(x)
       valorFinal = min(valorFinal, 1); // Escolhe o menor valor entre 1 e os outros valores
calculados
     else // Se elemento(x) > imagem(x)
       valorFinal = min(valorFinal, fuzzyImagem(l,c)); // Escolhe menor valor entre os outros
valores calculados e o pixel da imagem
     end

     // Definição dilatação Godel
      valorFinal = max(valorFinal, min(fuzzyElemento(l,c),fuzzyImagem(l,c))); //Escolhe o
maior valor entre os outros valores calculados e o menor valor entre o elemento(x) e a
imagem(x)

     // Definição erosão Goguen
     if fuzzyElemento(l,c) <= fuzzyImagem(l,c) // Se elemento(x)<= imagem(x)
        valorFinal = min(valorFinal, 1);  // Escolhe o menor valor entre 1 e os outros valores
calculados
     else // Se elemento(x) > imagem(x)
        valorFinal = min(valorFinal, fuzzyImagem(l,c)/fuzzyElemento(l,c));//Escolhe menor
valor entre os outros valores calculados e a divisão da imagem(x) pelo elemento(x)
     end

     // Definição erosão Goguen
     valorFinal = max(valorFinal, fuzzyImagem(l,c).*fuzzyElemento(l,c)); //Escolhe o maior
valor entre os outros valores calculados e a multiplicação do elemento(x) pela imagem(x)

    // Definição erosão Lukaciewisz
    valorFinal = min(valorFinal, min(1, 1 - fuzzyElemento(l,c)+fuzzyImagem(l,c))); // Escolhe
o menor entre os outros valores e 1 menos a soma do elemento(x) mais a imagem(x)

    // Definição erosão Lukaciewisz
    valorFinal = max(valorFinal, max(0, fuzzyElemento(l,c)+fuzzyImagem(l,c) - 1)); // Escolhe
o maior entre os outros valores e a soma do elemento(x) mais a imagem(x) menos 1
```

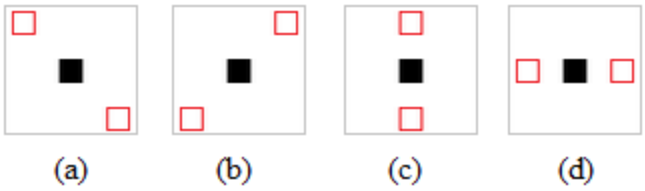
O **trecho de código 3** acima mostra a codificação das definições de dilatação e erosão para as implicações usadas neste trabalho.

4.1.3 Máscara de corte

O algoritmo máscara de corte foi desenvolvido com a finalidade de remover pequenos pontos e linhas estreitas dentro de imagens binárias. Seu conceito foi baseado nas definições de vizinhança $N_8(p)$.

A máscara funciona analisando a adjacência de cada pixel para decidir se ele pertence a uma grande estrutura ou se ele é apenas um pequeno ponto ou traço estreito. A analise é realizada verificando-se os *pixels* nos extremos das retas diagonais, vertical e horizontal a uma determinada distância de um pixel central. Caso em uma destas retas o pixel central seja preto e os *pixels* nos extremos sejam brancos o pixel central é convertido para branco. A **Figura 6** abaixo ilustra os pontos de análise do algoritmo para uma distância 1.

Figura 6: (a) analise diagonal 1, (b) analise diagonal 2, (c) analise vertical, (d) analise horizontal.

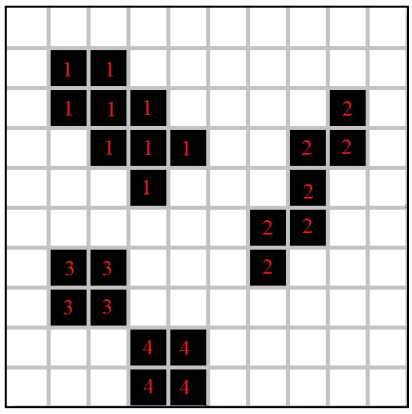


4.1.4 Códigos de contagem de objetos

A função de contagem de objetos tem como objetivo contar a quantidade de objetos e calcular seu tamanho em pixel (quantidade de pixel do objeto) dentro de uma imagem binária.

O algoritmo foi implementado seguindo as definições de vizinhança $N_4(p)$ onde apenas os vizinhos horizontais e verticais são considerados. O algoritmo percorre toda a matriz linha por linha e quando um pixel preto é encontrado ele recebe um número que será seu identificador como objeto e os *pixels* vizinhos a este também receberam o mesmo numero. A **Figura 7** abaixo exemplifica essa marcação do algoritmo.

Figura 7: imagem representa a marcação do algoritmo de contagem.

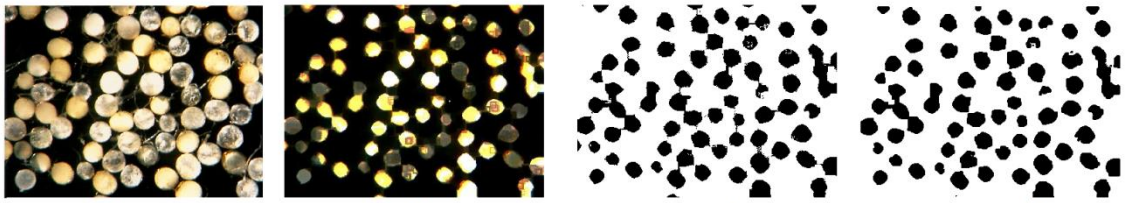


Após a marcação, a quantidade de objetos será o número de identificação do último objeto marcado e o tamanho de cada objeto é calculado somando os *pixels* marcados com mesmo identificador. Estes dois valores são importantes para se estimar o tamanho médio dos esporos.

4.2 Processo

O processo de quantificação de esporos de fungo por meio de fotos digitais se divide em duas partes: a primeira visa separar os esporos na imagem por meio de operadores morfológicos, transformação para imagem binária e uso de mascara; a segunda consiste em contar esporos separados dentro da imagem. A **Figura 8**, abaixo, mostra todas as etapas do processo de quantificação desde a separação até a imagem usada para contagem.

Figura 8: (a)imagem original, (b) imagem processada pelos operadores, (c) imagem convertida para binário, (d) imagem processada pela mascara de corte.



A imagem original (a) é processada por meio de operadores morfológicos a fim de separar os esporos que estejam juntos, em seguida, a imagem resultante (b) é convertida para binária (c) na qual o fundo se torna branco e os possíveis esporos ficam pretos, novamente a imagem é processada para retirada de finas linhas que ainda liguem esporos vizinhos e pequenos pontos que tenham surgido durante o processo. A imagem resultante (d) de todo esse processo é a imagem que passa pelo processo de contagem.

4.2.1 Processo de separação

O processo de separação consiste no uso de operadores morfológicos *fuzzy* e outros algoritmos auxiliares que tem como finalidade afastar objetos próximos dentro da imagem. As

três variáveis principais do processo são: o operador morfológico, a implicação/conjunção e o elemento estruturante.

A seguir são mostrados os passos tomados para definir estas variáveis.

4.2.1.1 Testes dos operadores

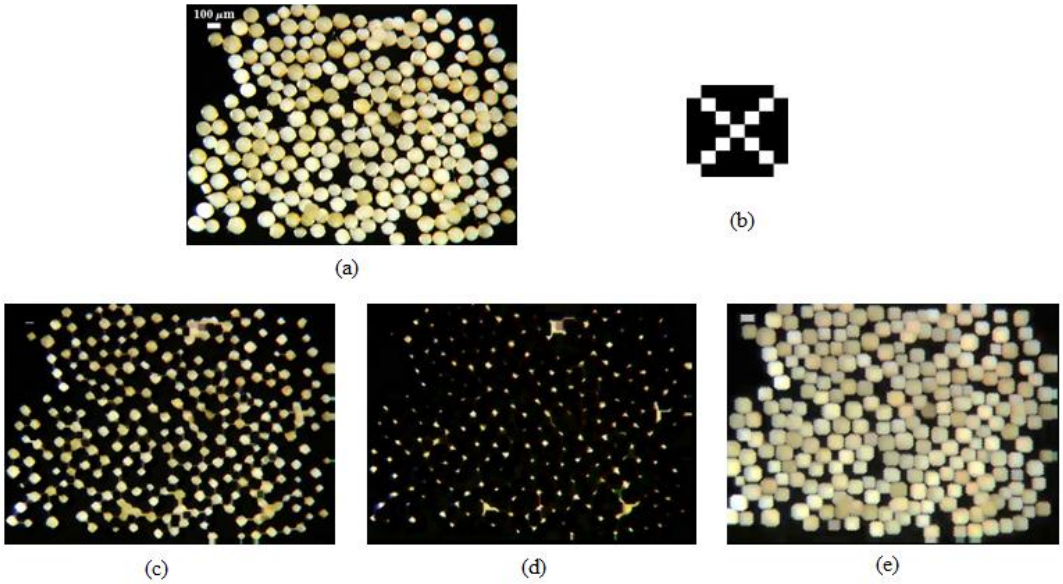
Para a escolha do operador foi montada uma tabela com todas as possibilidades de operações possíveis (18 combinações) tais possibilidades são mostradas na tabela abaixo.

Tabela 1: Tabela de operações possíveis

N	Operador	Implicações/ conjunções	Função de normalização
1	Erosão	Godel	x/255
2	Erosão	Godel	seno(x)
3	Erosão	Godel	tangente(x)
4	Erosão	Goguen	x/255
5	Erosão	Goguen	seno(x)
6	Erosão	Goguen	tangente(x)
7	Erosão	Lukaciewisz	x/255
8	Erosão	Lukaciewisz	seno(x)
9	Erosão	Lukaciewisz	tangente(x)
10	Dilatação	Godel	x/255
11	Dilatação	Godel	seno(x)
12	Dilatação	Godel	tangente(x)
13	Dilatação	Goguen	x/255
14	Dilatação	Goguen	seno(x)
15	Dilatação	Goguen	tangente(x)
16	Dilatação	Lukaciewisz	x/255
17	Dilatação	Lukaciewisz	seno(x)
18	Dilatação	Lukaciewisz	tangente(x)

O operador morfológico de erosão tem como característica diminuir a imagem e o operador de dilatação aumentar, desta forma, podemos utilizar a erosão para separar os esporos e a dilatação para uma possível reconstrução de esporos pequenos. Para visualizar como estas características se comportam nas imagens dos esporos uma sequência de ate três processamentos seguidos foram realizados em uma imagem com todas as combinações de modos de operadores da **Tabela 1** o que resultou em 6.175 imagens. Abaixo a **Figura 9** mostra alguns dos resultados obtidos com o processo.

Figura 9: (a) esporos fungo *claroidwh,* (b) elemento estruturante, (c) resultado operação modo 1, (d) resultado operação em sequencia modo 4-2, (e) resultado operação em sequencia modo 4-16-18.



Nas imagens resultantes foi possível observar que os operadores se comportam como o esperado onde o operador de erosão diminuiu os esporos e o de dilatação os aumentou. Desta forma, fica claro que o uso do operador de erosão é ideal para a separação dos esporos, no entanto, a eficácia do operador de dilatação para reconstrução de pequenos esporos fica reduzida já que ao mesmo tempo em que aumenta estes esporos acaba juntando novamente esporos separados.

4.2.1.2 Escolha da implicação/conjunção

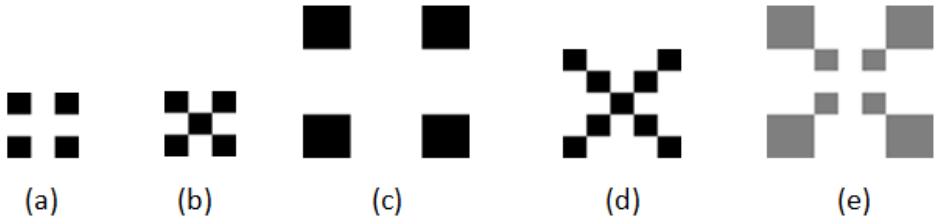
Com a confirmação das características dos operadores nas imagens de fungos o passo seguinte deve ser determinar qual das implicações/conjunções separam melhor. Para escolha da melhor implicação/conjunção, novos testes foram realizados em 37 imagens retiradas do site: *invam.caf.wvu.edu*, nestes testes são levados em conta o quanto o processo separa os fungo e qual a porcentagem de acerto na contagem. Cinco testes foram realizados utilizando os operadores e x/255 como função de normalização. A contagem foi realizada utilizando o método de contagem de objetos em imagem binária e a aproximação pela media. Estes métodos serão explicados mais a frente.

Os 5 testes foram realizados da seguinte forma:

- Teste 1- sequencia de cinco erosões e uma dilatação utilizando um elemento de 3x3pixel.
- Teste 2- sequencia de quatro erosões e uma dilatação utilizando um elemento de 3x3pixel.
- Teste 3- sequencia de duas erosões e uma dilatação utilizando um elemento de 7x7pixel.
- Teste 4- sequencia de quatro erosões e uma dilatação utilizando um elemento de 5x5pixel.
- Teste 5- sequencia de três erosões e uma dilatação utilizando um elemento de 7x7pixel.

A **Figura 10** abaixo mostra os elementos usados nos testes.

Figura 10: (a) elemento usado no teste 1, (b) elemento usado no teste 2, (c) elemento usado no teste 3, (d) elemento usado no teste 4, (e) elemento usado no teste 5.



A partir dos testes foi possível concluir que a implicações/conjunções de Gödel separou melhor os esporos e, assim, obteve-se melhor resultados na contagem.

Analisando a quantidade de operações e elemento estruturante, pode se fazer uma relação em que a *quantidade de operações é inversamente proporcional ao tamanho do elemento*, pois nos casos em que o elemento usado foi maior e a quantidade de operações foi menor o resultado ainda se manteve próximo dos demais. Já a mudança na cor do elemento mostra que *quanto mais clara a cor, maior o número de operações* necessárias para separar os fungos de forma satisfatória. **A Tabela 2** abaixo mostra quantas das imagens tiveram acerto acima de 80% para cada teste realizado.

Tabela 2: Resultados dos testes implicação/conjunção

Numero de imagens com contagem acima de 80% de acerto sobre um conjunto de 37 imagens			
	Implicação/Conjunção	Contagem simples	Contagem media
Teste 1	Godel	23	28
Teste 1	Gogue	10	21
Teste 1	Lukaciewisz	13	8
Teste 2	Godel	14	24
Teste 2	Gogue	5	18
Teste 2	Lukaciewisz	13	10
Teste 3	Godel	12	26
Teste 3	Gogue	12	26
Teste 3	Lukaciewisz	13	24
Teste 4	Godel	22	25
Teste 4	Gogue	24	23
Teste 4	Lukaciewisz	23	15
Teste 5	Godel	11	9
Teste 5	Gogue	16	17
Teste 5	Lukaciewisz	2	2

4.2.1.3 escolha do elemento e uso da mascara de corte de linha.

Durante os testes para escolha da implicação/conjunção o elemento estruturante mostrou se de vital importância para o processo de separação e agora focaremos em sua escolha. A partir deste ponto a mascara de corte de linhas foi implementada e incluída no processo e assim os fungos que ficam juntos por linhas depois do processamento por operadores e conversão para binária podem ser separado por esta mascara.

Para escolha do parâmetro de distância utilizado na mascara as imagens binárias resultantes do processo de escolha da implicação/conjunção foram processadas com parâmetros de distancia de 1 a 4. A partir das analises visuais dos resultados o parâmetro de distância 2 se mostrou mais eficiente apagando a maioria das linhas e não apagando muitos fungos que no processo de separação ficaram muito pequenos.

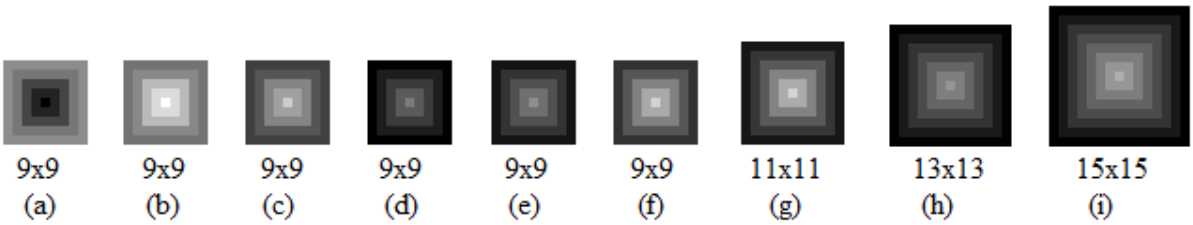
Sobre a escolha do elemento sabemos que aumentando o tamanho do elemento o numero de erosões pode ser diminuído então os testes foram feitos com apenas uma erosão o que reduziu de 2 a 4 vezes o tempo de execução para o processo de contagem. A dilatação foi removida do processo, pois este operador era usado para recuperar a forma características dos fungos após as erosões e, no entanto não se mostrou eficaz para a contagem tendo em vista que ele voltava a juntar os fungos.

Nove elementos de tamanho entre 9X9 e 15X15 *pixels* e cores variando dentro da escala de cinza foram usados. **A Tabela 3** abaixo mostra o resultado dos testes e a **Figura 11** mostra os elementos usados.

Tabela 3: Resultados dos testes com os elementos

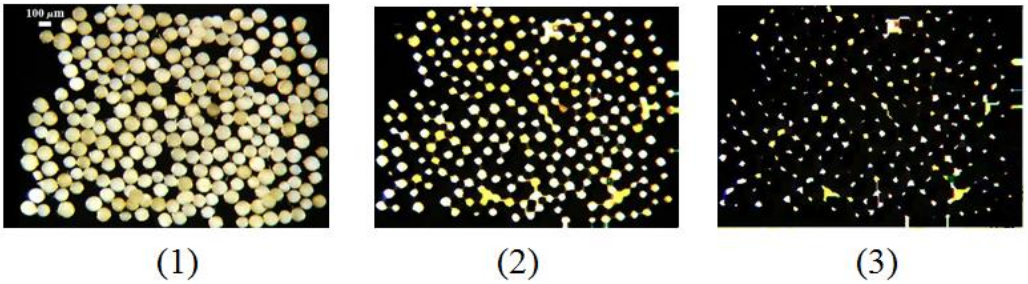
Numero de imagens com contagem acima de 80% de acerto sobre um conjunto de 37 imagens				
	Cont. simples	Cont. media	Cont. pós mascara	Cont. media pós mascara
Elemento (a)	18	19	22	22
Elemento (b)	16	19	23	23
Elemento (c)	13	12	19	21
Elemento (d)	6	6	4	8
Elemento (e)	10	9	11	15
Elemento (f)	12	13	18	20
Elemento (g)	14	15	17	21
Elemento (h)	7	10	17	19
Elemento (i)	6	8	18	19

Figura 11: (a,..., i) elementos usados nos testes para escolha de elemento.



O uso de elementos mais escuros ou muito grandes acabou se mostrando pouco eficaz para separar os fungos, tais elementos acabaram apagando os fungos na imagem ou tonando-os muito pequenos, o que impossibilitou a sua contagem, pois a máscara pode apaga-los. A **Figura 12** abaixo mostra a diferença de resultado entre dois elementos. O elemento(b) pequeno e claro e o elemento (i) grande e escuro.

Figura 12: (1) Imagem original, (2) Resultado com elemento(b), Resultado com elemento (i).



Ao fim deste processo o elemento (b) foi escolhido como melhor opção para a contagem e a imagem do elemento foi transformada em uma matriz e acrescentada ao código como uma constate do processo.

4.2.1.3 processo de separação finalizado

Com a escolha do operador, da implicação/conjunção e do elemento o processo de separação pode ser concluído com a inclusão da conversão para binário e da mascara de corte ficando o processo configurado da seguinte maneira:

1- Utilizar erosão Gödel com elemento(b).
2- Converter para binário.
3- Utilizar mascara de corte com parâmetro de distância 2.

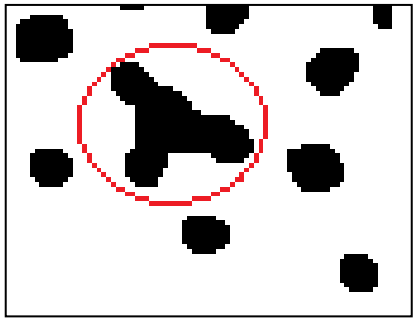
4.2.2 Processo de contagem

O processo de contagem consiste em contar objetos em uma imagem binária. No caso deste trabalho os objetos serão os esporos.

4.2.2.1 Processo de contagem simples

O processo de contagem simples faz apenas a contagem dos objetos na imagem utilizando o algoritmo de contagem sem dar importância ao tamanho de cada objeto, neste caso um objeto de 1 *pixels* de tamanho e outro com 100pixel de tamanho tem o mesmo peso na contagem. A **Figura 13** abaixo mostra o problema deste tipo de contagem onde o objeto circulado em vermelho tem o mesmo peso dos demais, no entanto, o objeto maior corresponde a 4 esporos que não foi possível separar no processo.

Figura 13: Fragmento de imagem de esporos de fungos claroidwh após o processo de separação.



4.2.2.2 Processo de contagem média

No processo de contagem simples ao não levar em conta o tamanho do objeto esporos que ficaram juntos ou pedaços pequenos de algum esporo que se separou são contados de forma errônea por esse motivo uma verificação desses objetos se faz necessária.

Para descobrir se um objeto representa um único, vários ou um pequeno pedaço de esporo foi feito um calculo médio do tamanho destes objetos utilizando a quantidade de *pixels* que cada um tem. Considerando que vários esporos únicos foram separados o calculo médio terá valor próximo ao tamanho do esporo.

A partir da media M e de cada conjunto Q, a quantidade de esporos P é calculada com da seguinte forma:

Primeiro calcula se a media M do tamanho dos objetos (esporos)

M= Quantidade de *pixels* pretos / quantidade de objetos

Segundo calcula se os esporos P1 inteiros descartando o resto da divisão.

P1=(Q)/M (divisão inteira sendo R o resto)

Terceiro calcula se o resto R da divisão.

R=(Q Modulo M) (resto da divisão)

Quarto calcula se, se o resto corresponde a um esporo menor que a media.

P2=(R+(M*7/100) /M) (soma aqueles que são superiores a 93% da media)

Por ultimo soma se os dois valores.

P=P1+P2;

Com este recurso a contagem obteve melhores resultado, o que pode ser observado nas tabelas, **Tabela 2** e **Tabela 3**.

5 Resultados e discussão

Com a finalização do processo de separação e contagem faremos agora algumas discussões sobre os resultados obtidos.

O processo de separação se mostrou satisfatório na maioria das fotos utilizadas nos testes, entretanto, em algumas delas a separação foi ineficiente. A falta de um padrão de cores e de tamanho dos esporos foi o fator mais relevante que afetou negativamente o processo de separação uma vez que um mesmo elemento estruturante traz resultados diferentes para padrões diferentes. Uma alternativa para aumentar a eficiência do processo de separação seria a escolha de um elemento estruturante para cada tipo de fungo.

No que tange o processo de contagem a contagem simples não se mostra muito eficiente tendo em vista que conta objetos de tamanhos muito diferentes atribuindo lhes o mesmo valor já a contagem media é uma melhoria da outra e ao utilizar a média do tamanho dos objetos (esporos) para calcular o valor de cada na contagem traz uma significativa melhora. Esse método, no entanto, é bastante influenciado pela separação, pois para que a media se aproxime do real uma boa quantidade de esporos devem estar separadas.

Como resultados das implementações um toolbox para a contagem foi desenvolvido e seu código acompanha este trabalho.

6 Conclusões e trabalhos futuros

Uma das principais aplicações do processamento digital de imagens e a contagem de objetos em uma imagem. A variedade de problemas de contagem de objetos é muito grande e vão desde células, bactérias até arvores, frutas, animais e pessoas (BARBEDO 2012). A limiarização, detecção de borda, erosão, dilatação são algumas das técnicas utilizadas ao longo do tempo no processo de contagem de objetos.

Este trabalho apresentou um método de quantificação de esporos de fungos micorrizicos, utilizando morfologia matemática *fuzzy* para separação dos esporos e algoritmos para contagem. Verificou-se que os operadores morfológicos *fuzzy* se mostraram eficientes para o processo de separação dos fungos e que o processo de contagem automática pode ser uma forma alternativa à contagem manual tendo em vista que o processo alcançou acertos acima de 80%.

Todos os algoritmos foram desenvolvidos na ferramenta Scilab que é um software livre de código aberto o que pode facilitar a utilização do método por outros desenvolvedores e pesquisadores.

Como trabalho futuro recomenda se a criação de métodos de contagem para cada espécie de fungo especifica o que tornaria a contagem ainda mais eficiente. Há ainda a possibilidade de extensão do método para outros objetos que tenham características similares as dos esporos, por exemplo, alguns tipos de células, bactérias, frutas e grãos.

Por último, a implementação de uma interface gráfica facilitaria a utilização do método por usuários comuns.

Referências

AGUADO, A.G., CANTANHEDE, M.A. Lógica Fuzzy. 2010. Disponível em: < http://www.ft.unicamp.br/liag/wp/monografias/monografias/2010_IA_FT_UNICAMP_logicaFuzzi.pdf>.Acesso em: 19 de ago. 2014.

ANDRADE, A.O., R.M.P. TRINDADE, D.S.MAIA, R.H.N.SANTIAGO & A. M. G. GUEREIRO (2012), Uso da morfologia matemática fuzzy na contagem esporos de fungos micorrízicos, *em* 'Recentes Avanços em Sistemas Fuzzy', II Congresso Brasileiro de Sistema Fuzzy.

ANDRADE, A.O. Um Sistema de Contagem baseado em Morfologia Matemática Fuzzy. Tese de doutorado (Ciências) Universidade Federal Do Rio Grande Do Norte, Natal. 2014.

BARBEDO, J.G.A. (2012), *Estado da Arte das Técnicas de Contagem de Elementos Específicos em Imagens Digitais*, Embrapa.

BERBARA, R.L.L. & FONSECA, H.M.A.C. Colonizacão e esporulacão de fungos micorrízicos arbusculares *in vitro*. In: SIQUEIRA, J.O., ed. Avanços em fundamentos e aplicação de micorrizas. Lavras, Universidade Federal de Lavras, 1996. p.39-65.

BOUCHET A., PASTORE J. e BALLARIN V. Segmentation of Medical Images using Fuzzy Mathematical Morphology. JCS T Vol. 7 No. 3. 2007. Disponível em: <http://www.fmi.uni-sofia.bg/courses/biomed/morphgeom/papers/JCST.pdf>. Acesso em: 15 ago. 2014.

BRITO,V.N. :Fungos Micorrízicos Arbusculares e Adubação Fosfatada Na Produção De Mudas De Paricá Universidade Estadual do Norte Fluminense – RJ JULHO – 2013. Disponível em: < http://uenf.br/Uenf/Downloads/PRODVEGETAL_3434_1381832041.pdf>. Acesso em: 13 jan. 2014.

COX, Earl. The fuzzy systems handbook: a practitioner's guide to building, using, and maintaining fuzzy systems . New York: AP Professional, 1994. Disponível em: <http://www.ebah.com.br/content/ABAAAfJPkAE/literatura-cox-the-fuzzy-systems-handbook>. Acessado em: 10 de ago. 2014.

FILHO,D.G. SCILAB 5.X . Disponível em: <http://euler.mat.ufrgs.br/~giacomo/Manuais-softw/SCILAB/Apostila%20de%20Scilab%20-%20atualizada.pdf>. Acessado em 13 de ago. 2014.

GASPARIN, J.P., A. BOUCHET, G. ABRAS, V. BALLARIM & J.I. PASTORE (2011), 'Medical image segmentation using the hsi color space and fuzzy mathematical morphology', *Journal of Physics* 332. Disponível em: < http://iopscience.iop.org/1742-6596/332/1/012033/pdf/1742-6596_332_1_012033.pdf> . Acesso em: 25 ago. 2014.

GERDEMAN J.E. and NICOLSON T.H.. Spores of mycorrhizal endogone species extracted from soil by wet sieving and decanting. *Trans. Br. mycol.Soc.*, 46:235-244, 1963. Disponível em:< http://www.sciencedirect.com/science/article/pii/S0007153663800790 > Acessado em: 18 ago. 2014.

GONZALES, R. C.; WOODS, R. E. Digital Image Processing. 2ª edição. Upper Saddle River: Prentice Hall. 2002. Disponível em: <http://users.dcc.uchile.cl/~jsaavedr/libros/dip_gw.pdf >. Acesso em: 18 novembro. 2014.

GOTO B.T e MAIA L.C.. Glomerospores: A new denominatio for the spores of glomeromycota, a group molecularly distinc from the zygomycota. *Mycotaxon*, (96):129-132, 2006. Disponível em: < file:///C:/Users/alecio/Downloads/Goto%20et%20al%202013_Acaulospora_endographis%20(1).pdf> . Acessado em: 18 ago. 2014.

JAKOBSEN, I.; SMITH, S.E. & SMITH, F.A. Function and diversity of arbuscular mycorrhizae in carbon and mineral nutrition. In: van der HEIJDEN, M.G.A. & SANDERS, I., eds. Mycorrhizal ecology. Berlin, Springer-Verlag, 2002. p.75-92.

(Ecological Studies, 157). Disponível em: <http://link.springer.com/chapter/10.1007/978-3-540-38364-2_3>. Acessado em: 18 ago. 2014.

JENKINS,W.R.. A rapid centrifugal-otation technique for separating nematodes from soil. *Plant Disease Report*, 8:692, 1964.

KLIR, George J; YUAN, BO Fuzzy Sets and Fuzzy Logic: Theory and Applications. New Jersey: Prentice Hall PTR, 1995. Disponível em: <http://is.iiita.ac.in/study/Computational%20Intelligence/klir_youn.pdf>. Acessado em: 10 de ago. 2014.

MATTA,W.N. Metodologia para Detecção de Máculas em Micrografias Utilizando Morfologia Matemática. Tese de Mestrado (Ciência da Computação) Universidade Federal de Minas Gerais. Belo Horizonte. 1998. Disponível em: <http://www.dcc.ufmg.br/pos/cursos/mestrado_defesas_detalhes.php?aluno=661>. Acesso em: 13 de ago. 2014.

MEN, H.; WU, Y.; LI, X.; KOU, Z.; YANG, S. Counting method of heterotrophicbacteria based on image processing. In: IEEE Conference on Cybernetics and Intelligent Systems, 7., 2008, Chengdu. *Proceedings*… London: IEEE: Piscataway, 2008. p. 1238–1241.

MOREIRA, F.M.S., SIQUEIRA, J.O. (2006) Microbiologia e bioquímica do solo. Lavras: UFLA, p. 729.

MORFOLOGIA In: WIKIPÉDIA: The Free Encyclopedia. Disponível em< http://pt.wikipedia.org/wiki/Morfologia>. Acesso em: 13 de ago. 2014.

ORTEGA, N.R.S. *Contribuição do Aplicação da Teoria de Conjuntos Fuzzy a Problemas da Biomedicina*. Tese de Doutoramento (ciências) Instituto de Física da Universidade São Paulo, São Paulo. 2001. Disponível em: <https://www.ime.usp.br/~tonelli/verao-fuzzy/neli/principal.pdf>. Acesso em: 19 de ago. 2014.

POOMCOKRAK, J. & C. NEATPISARNVANIT (2008), 'Red blood cells extraction and counting.',*International ymposium on Biomedical Engineering*, 3(199-203). Porter, W.M. (1979), 'The"most

REDECKER D., SCHUBLER A., STOCKINGER H., STUMER S.L., MORTON J.B., and WALKER C. An evidence-based consensus for the classification of arbucular mycorrhizal fungi (glomeromycota). *Mycorrhiza*, 2013. Disponível em:< http://books.google.com.br/books?id=2LPrAgAAQBAJ&pg=PA288&lpg=PA288&dq=An+evidence-ased+consensus+for+the+classification+of+arbuscular+mycorrhizal+fungi&source=bl&ots=aFqvi-ooJH&sig=-GYDX7JdzTdA-fybhCeJJPo6KqM&hl=pt-BR&sa=X&ei=nVV3VOiJKYejNsfng-AC&ved=0CHUQ6AEwCQ#v=onepage&q=An%20evidence-based%20consensus%20for%20the%20classification%20of%20arbuscular%20mycorrhizal%20fungi&f=false >. Acessado em: 18 ago. 2014.

RIBEIRO, S.F. Um Sistema de Visão Inteligente para Detecção e Reconhecimento de Peças em um Tabuleiro de Xadrez em Tempo Real. Tese mestrado (Informática) Universidade Federal da Paraíba, Campina Grande. 2001. Disponível em : <http://docs.computacao.ufcg.edu.br/posgraduacao/dissertacoes/2001/Dissertacao_SergioFaustinoRibeiro.pdf>. Acesso em: 13 de ago. 2014.

SALIS, T.T. & G.A.S. PEREIRA. (2006), Contagem automática de tarugos de aço por meio de visão computacional., Dissertação de mestrado, Universidade Federal de Minas Gerais. Disponível em: <http://www.ppgee.ufmg.br/documentos/PublicacoesDefesas/786/ContagemAutomatica.pdf> . Acesso em: 25 ago. 2014.

SIQUEIRA,J.O., LAMBAIS,M.R.S., STUMER,L.: Fungo Micorrízicos Arbusculares. Biotecnologia Ciência & Desenvolvimento - nº 25- março/abril 2002. Disponível em: < http://www.biotecnologia.com.br/revista/bio25/fungos.pdf>. Acesso em: 13 jan. 2014.

SOUZA,F.A., STUMER, S.L., CARRENHO R., and TRUFEM,S.F.B. *Micorrizas: 30 anos de pesquisas no Brasil*, chapter Classificação e taxonomia de fungos micorrízicos arbusculares e sua diversidade e ocorrência no Brasil, pages 15-73. Editora UFLa, 2010.

SOUZA, L.A.B., BONNASSIS, P.A.P., SILVA FILHO, G.N., OLIVEIRA, V.L. (2008) Novos isolados de fungos ectomicorrízicos e o crescimento de eucalipto. Pesquisa Agropecuária Brasileira, p. 235-24.

WANGENHEIM,A.D. Introdução a Visão Computacional. Capitulo de Morfologia Matemática. Disponível em: <http://www.inf.ufsc.br/~visao/morfologia.pdf>. Acessado em 13 de ago. 2014.

VON ALTROCK, Constantin. Fuzzy logic and neuroFuzzy applications in busines and finance. New Jersey: Prentice Hall PTR, 1996.

ZADEH, L. (1965). Fuzzy Sets - *Information and Control*, vol. 8, pp 338-353. Disponível em: <http://www.cs.berkeley.edu/~zadeh/papers/Fuzzy%20Sets-Information%20and%20Control-1965.pdf>. Acesso em: 10 de ago. 2014.

Apêndice - código método de contagem de esporos de fungos

```
//função para contagem de fungos
//recebe como parametro um imagem (não o caminho )
//utiliza o alxilio de outra funcão que faz a contagem ContaFungo

function [qt1, qtm1, qt, qtm]=contaFungos(imagem)
   elemento=ones (9,9,3);
   elemento=[
   115  115  115  114  115  114  118  114  115;
   115  139  141  142  136  143  140  139  115;
   117  139  185  183  186  181  186  142  115;
   114  138  182  224  219  220  189  136  115;
   120  139  188  216  255  219  186  142  115;
   111  142  183  222  220  218  185  140  115;
   116  139  184  190  184  185  186  138  115;
   113  142  141  136  141  143  137  142  115;
   115  115  115  115  115  115  115  115  115];
   elemento(:,:,2) =[
   115  115  115  114  115  114  118  114  115;
   115  139  141  142  136  143  140  139  115;
   117  139  185  183  186  181  186  142  115;
   114  138  182  224  219  220  189  136  115;
   120  139  188  216  255  219  186  142  115;
   111  142  183  222  220  218  185  140  115;
   116  139  184  190  184  185  186  138  115;
   113  142  141  136  141  143  137  142  115;
   115  115  115  115  115  115  115  115  115];
   elemento(:,:,3) =[
   115  115  115  114  115  114  118  114  115;
   115  139  141  142  136  143  140  139  115;
   117  139  185  183  186  181  186  142  115;
   114  138  182  224  219  220  189  136  115;
   120  139  188  216  255  219  186  142  115;
   111  142  183  222  220  218  185  140  115;
   116  139  184  190  184  185  186  138  115;
   113  142  141  136  141  143  137  142  115;
   115  115  115  115  115  115  115  115  115];

   imagemProcessada=processamento( imagem,elemento, 1, 1 ,1,1,1, 1);
   imagemFundoBranco=converteBinaria(imagemProcessada);
   [qt1 qtm1 t]=contaEstrutura(imagemFundoBranco);
   imagemMascara= mascara2(2,imagemFundoBranco);
   [qt  qtm  t]=contaEstrutura(imagemMascara);

endfunction

// função que converte a imagem para binaria
function matrizResultante=converteBinaria(matrizImagem)
matrizImagem=uint16(matrizImagem);
```

```
linhasImagem = size(matrizImagem,1);
colunasImagem = size(matrizImagem,2);

matrizResultante = (ones(linhasImagem, colunasImagem, 3));
matrizMediaDosPixels =(ones(linhasImagem, colunasImagem, 3));
matrizMediaDosPixels = (matrizImagem(:,:,1) +  matrizImagem(:,:,2) + matrizImagem(:,:,3))/3;

maiorValor = max(max(matrizMediaDosPixels));
menorValor = min(min(matrizMediaDosPixels));

valorDeCorte = (maiorValor + menorValor)/2;

resposta=matrizMediaDosPixels<valorDeCorte;

matrizResultante=resposta*255;

endfunction

// esta função tenta apagar linhas finas que ligão fungos
// os parametros de entrada são a largura da linha e o caminho de onde esta
// a imagem

function [matrizImagem]=mascara2(distancia, matrizImagem)

//matrizImagem = imread(caminho);

linhasImagem = size(matrizImagem,1);
colunasImagem = size(matrizImagem,2);

matrizAuxiliar = ones(linhasImagem+(distancia+8)*2, colunasImagem+(distancia+8)*2);
matrizAuxiliar=matrizAuxiliar*255;
matrizAuxiliar(distancia+8:linhasImagem+distancia+7, distancia+8:colunasImagem+distancia+7) =
matrizImagem(:,:,1);

   for l=distancia+8:linhasImagem+distancia+8
      for c=distancia+8:colunasImagem+distancia+8
         pixel=0;
         if matrizAuxiliar(l,c)==0
           for d=0:distancia
             [pixel, matrizAuxiliar]=diagonal1(l,c,matrizAuxiliar,distancia-d);
             if pixel>0
               break;
             end
           end
         end
         pixel=0;
         if matrizAuxiliar(l,c)==0
           for d=0:distancia
             [pixel, matrizAuxiliar]=diagonal2(l,c,matrizAuxiliar,distancia-d);
             if pixel>0
```

```
            break;
          end
        end
      end
      pixel=0;
      if matrizAuxiliar(l,c)==0
        for d=0:distancia
          [pixel, matrizAuxiliar]=vertical(l,c,matrizAuxiliar,distancia-d);
          if pixel>0
            break;
          end
        end
      end
      pixel=0;
      if matrizAuxiliar(l,c)==0
        for d=0:distancia
          [pixel, matrizAuxiliar]=horizontal(l,c,matrizAuxiliar,distancia-d);
          if pixel>0
            break;
          end
        end
      end

    end
  end

matrizImagem=matrizAuxiliar(distancia+8:linhasImagem+distancia+7,
distancia+8:colunasImagem+distancia+7);
endfunction

//função auxiliar para mascara
function [pixel, matrizAuxiliar]=diagonal1(linha, coluna, matrizAuxiliar, distancia)

  pixel=0;
  falhou=1;

  tamanho=abs(distancia*2);
  x=0;//caso distancia zero
  if distancia==0
    tamanho=1;
  end
  for i=1:tamanho
    if matrizAuxiliar(linha-i,coluna-tamanho+i-1)==0 | matrizAuxiliar(linha+tamanho+1-
i,coluna+i)==0
      falhou=0;
      break;
    end
  end
  if falhou ==1
```

```
      for x=0:distancia*2
        matrizAuxiliar(linha-distancia+x,coluna-distancia+x)=255;
      end
      pixel=255;

    end

endfunction

//função auxiliar para mascara
function [pixel, matrizAuxiliar]=diagonal2(linha, coluna, matrizAuxiliar, distancia)

   pixel=0;
   falhou=1;

   tamanho=abs(distancia*2);
   x=0; //caso distancia zero
   if distancia==0
      tamanho=1;
   end
   for i=1:tamanho
    if matrizAuxiliar(linha+i,coluna-tamanho+i-1)==0 | matrizAuxiliar(linha-tamanho-
1+i,coluna+i)==0
        falhou=0;
        break;
     end
   end
   if falhou ==1

      for x=0:distancia*2
        matrizAuxiliar(linha+distancia-x,coluna-distancia+x)=4;
      end
      pixel=255;

    end

endfunction

//função auxiliar para mascara
function [pixel, matrizAuxiliar]=vertical(linha, coluna, matrizAuxiliar, distancia)

   pixel=0;
   falhou=1;

   tamanho=abs(distancia*2-1);
   x=0; //caso distancia zero
   if distancia==0
      x=1;
   end
   for i=0:tamanho-1
```

```
     if matrizAuxiliar(linha-distancia-1,coluna-distancia-x+i+1)==0 |
matrizAuxiliar(linha+distancia+1,coluna-distancia-x+i+1)==0
        falhou=0;
        break;
     end
   end
   if falhou ==1
     for x=0:distancia*2
       matrizAuxiliar(linha-distancia+x,coluna)=255;
     end
     pixel=255;

    end

endfunction

//função auxiliar para mascara
function [pixel, matrizAuxiliar]=horizontal(linha, coluna, matrizAuxiliar, distancia)

   pixel=0;
   falhou=1;

   tamanho=abs(distancia*2-1);
   x=0; //caso distancia zero
   if distancia==0
      x=1;
   end
   for i=0:tamanho-1
     if  matrizAuxiliar(linha-distancia-x+i+1,coluna-distancia-1)==0 | matrizAuxiliar(linha-
distancia-x+i+1,coluna+distancia+1)==0
        falhou=0;
        break;
     end
   end
   if falhou ==1
     for x=0:distancia*2
       matrizAuxiliar(linha,coluna-distancia+x)=255;
     end
     pixel=255;

    end

endfunction

//função para contar objetos
function [numeroDeBolinhas, numerodebolinhasmedias, tempo]=contaEstrutura(matrizImagem)

   linhasImagem = size(matrizImagem,1);
   colunasImagem = size(matrizImagem,2);
```

```
    matrizAuxiliar = ones(linhasImagem+2, colunasImagem+2);

    matrizAuxiliar(2:linhasImagem+1, 2:colunasImagem+1) = matrizImagem(:,:,1); // Matriz auxiliar é a
matriz da imagem com uma borda de pixels brancos
    matrizAuxiliarValores = ones(linhasImagem+2, colunasImagem+2);
    matrizAuxiliarValores2 = ones(linhasImagem+2, colunasImagem+2);
    numeroDeBolinhas = 0; // Essa variável irá armazenar o numero total de 'formas' encontradas na
imagem
    jaFoiContada = -1;    // Essa variavel verifica se o pixel atual faz parte de uma 'forma' que já foi
contada
    linha = 2; // A matriz é percorrida a partir da linha dois para ignorar a borda da matriz auxiliar
    coluna = 2; // A matriz é percorrida a partir da coluna dois para ignorar a borda da matriz auxiliar
    valorDaForma = -1; // Cada forma tem um valor que é a posição do primeiro pixel encontrado na
forma
    vetorValoresLinhaAnterior = zeros(1,colunasImagem); // A linha atual pode estar ligada à varias
formas já contabilizadas
    // e esse vetor armazena o valor dessas formas
    posicaoVetorValores = 1; // Posição que percorre o vetor de valores a medida que novos valores de
formas são encontrados
    colunaInicialDaLinhaPreta = 0; // Variavel que armazena a coluna do primeiro pixel preto da forma na
linha atual da forma sendo analisada

    // Esses laços percorrem a matriz auxiliar
    while (linha <= linhasImagem+1)
        while (coluna <= colunasImagem+1)
            if (matrizAuxiliar(linha,coluna) == 0) // Verifica  se o pixel atual é preto
                colunaInicialDaLinhaPreta = coluna;
                jaFoiContada = 0; // A principio, o pixel preto encontrado faz parte de uma 'forma' que ainda
não foi contada
                while (matrizAuxiliar(linha,coluna) == 0 & coluna <= colunasImagem+1) // Enquanto houver
pixels pretos consecutivos...
                    if (matrizAuxiliar(linha-1,coluna) == 0) // Se há uma pixel preto vizinho ao analisado
atualmente, então o pixel faz parte de uma forma já analisada
                        jaFoiContada = 1; // O valor '1' para a variaval 'jaFoiContada' indica que o pixel pertence
a uma forma já contada
                        if (size(find(vetorValoresLinhaAnterior == matrizAuxiliarValores(linha-1,coluna)),2) ==
0);
                            vetorValoresLinhaAnterior(posicaoVetorValores) = matrizAuxiliarValores(linha-
1,coluna);
                            posicaoVetorValores = posicaoVetorValores+1;
                        end
                        if (valorDaForma == -1);
                            valorDaForma = matrizAuxiliarValores(linha-1,coluna);
                        end
                    end
                    coluna = coluna+1;
                end
            end
            if (jaFoiContada == 0) // Se 'jaFoiContada' for 0, então a forma encontrada ainda não foi
contabilizada
```

```
            numeroDeBolinhas = numeroDeBolinhas+1; // O numero de bolinhas (formas) é
incrementado
            valorDaForma =  numeroDeBolinhas;
            matrizAuxiliarValores(linha, colunaInicialDaLinhaPreta:coluna-1) = valorDaForma*ones(1,
coluna-colunaInicialDaLinhaPreta);
        end
        if (jaFoiContada == 1)
            matrizAuxiliarValores(linha, colunaInicialDaLinhaPreta:coluna-1) = valorDaForma*ones(1,
coluna-colunaInicialDaLinhaPreta);
            numeroDeBolinhas = numeroDeBolinhas-(posicaoVetorValores-2);

            // Esse laço percorre a linha anterior à atual substituindo os valores dos pixels que
            // pertencem à forma atual mas possuiam outro valor
            for c=2:colunasImagem+1
                valorPixel2 = matrizAuxiliarValores(linha-1,c);
                find(vetorValoresLinhaAnterior == valorPixel2);
                if (size((find(vetorValoresLinhaAnterior == valorPixel2)),2) ~= 0)
                    matrizAuxiliarValores(linha-1,c) = valorDaForma;
                end
            end
            // Esse laço percorre a linha atual substituindo os valores dos pixels que
            // pertencem à forma atual mas possuiam outro valor
            for c=2:coluna
                valorPixel = matrizAuxiliarValores(linha,c);
                find(vetorValoresLinhaAnterior == valorPixel);
                if (size((find(vetorValoresLinhaAnterior == valorPixel)),2) ~= 0)
                    matrizAuxiliarValores(linha,c) = valorDaForma;
                end
            end

        end
            jaFoiContada = -1; // O valor '-1' indica que nenhuma forma foi encontrada na linha
            coluna = coluna+1;
            valorDaForma = -1;
            vetorValoresLinhaAnterior = zeros(1,colunasImagem);
            posicaoVetorValores = 1;
            colunaInicialDaLinhaPreta = 0;
        end

        linha = linha+1;
        coluna = 2;
    end
    valores=[];
    quantidade=[];
    for l=1:linhasImagem+1
        for c=1:colunasImagem+1

            if(matrizAuxiliarValores(l,c)>1)

                posicao=find(valores==matrizAuxiliarValores(l,c));
```

```
            if(posicao>0)

               quantidade(posicao)=quantidade(posicao)+1  ;
            else

               valores=[valores matrizAuxiliarValores(l,c)] ;
               quantidade=[quantidade 1];
            end
         end
      end
   end
   //esta é a quantidade de estruturas encontradas
   quantidade;

   matrizAuxiliarValores;
   imagem=ones(size(matrizAuxiliarValores,1) ,size(matrizAuxiliarValores,2),3);

   //calcula a media do tamanho dos fungos

   media=sum(quantidade)/size(quantidade,2)
   disp(media);
   //exclui os fungos que são menores que metade da media
   qt=round((quantidade/media)-0.5)
   //qt2=rem(quantidade,media)
   qt2=quantidade - fix ( quantidade / media ) * media;
   //calcula a quantidade com o valor medio do tamanho dos fungos
   qt2=round((qt2+(media*30/100))/media)
   qt=qt+qt2
   numeroDeBolinhas; //quantidade de fungos sem a media
   numerodebolinhasmedias = sum(qt);//quantidade de fungos com a media
   //essa variavel tempo pode ser usada mostrar o tempo de processamente usando
   //a função tic toc
   tempo=0;

endfunction

// função processamento realiza as operações morfologicas de erosão e dilatação
// Os parâmetros dessa função são: - imagem = nome do arquivo da imagem
//                         - elementoEstruturante = nome do arquivo do elemento estruturante
//                         - linhaOrigemElemento = inteiro que representa o eixo X da origem do elemento
(valor mínimo = 1)
//                         - colunaOrigemElemento = inteiro que representa o eixo Y da origem do elemento
(valor mínimo = 1)
//                         - tipoProcessamento = inteiro que representa o tipo de procesamento
//                                        escolhido pelo usuário (2 = Erosao Fodor; 3 = Erosao Godel...)
//                         - tipoImagem = inteiro que representa o tipo da imagem selecionada pelo usuário
//                                  (0 = Escala de Cinza; 1 = Colorida)
function [imagem]=processamento(imagem, elementoEstruturante, linhaOrigemElemento,
colunaOrigemElemento, tipoProcessamento, tipoAdjuncao, tipoFuncao, tipoFuzzificacao)
```

```
   imagemRD=imread(imagem);
   //elementoEstruturanteRD=imread(elementoEstruturante);
   imagem=processamentoImagensColoridas(imagemRD, elementoEstruturante,
linhaOrigemElemento, colunaOrigemElemento , tipoProcessamento, tipoAdjuncao, tipoFuncao,
tipoFuzzificacao);

endfunction

function [matrizResultante2]=processamentoImagensColoridas(imagem, elementoEstruturante,
linhaOrigemElemento, colunaOrigemElemento, tipoProcessamento, tipoAdjuncao, tipoFuncao,
tipoFuzzificacao)

   matrizImagem = imagem; // matrizImagem é a matriz tridimensional que armazena os dados dos pixel
da imagem
   matrizElementoEstruturante = elementoEstruturante; // matrizElementoEstruturante é a matriz
tridimensional que armazena os dados dos pixel do elemento estruturante
   linhasElemento = size(matrizElementoEstruturante,1);
   colunasElemento = size(matrizElementoEstruturante,2);
   //matrizElementoEstruturante2 = ones (linhasElemento, colunasElemento,3);
   //substitui os pixels vermelhos para que eles adquiram um valor fora do intervalo [0,1]
   for i = 1: linhasElemento
      for j = 1:colunasElemento
         if ~(size(matrizElementoEstruturante ,3)>1 & matrizElementoEstruturante(i,j,1) ==
matrizElementoEstruturante(i,j,2) & matrizElementoEstruturante(i,j,2) ==
matrizElementoEstruturante(i,j,3))
            matrizElementoEstruturante2(i,j,1) = 892;
            matrizElementoEstruturante2(i,j,2) = 892;
            matrizElementoEstruturante2(i,j,3) = 892;
         else
            matrizElementoEstruturante2(i,j,1) = matrizElementoEstruturante(i,j,1);
            matrizElementoEstruturante2(i,j,2) = matrizElementoEstruturante(i,j,2);
            matrizElementoEstruturante2(i,j,3) = matrizElementoEstruturante(i,j,3);
         end
      end
   end
   matrizResultante(:,:,1) = processamentoImagensAuxiliar( matrizImagem(:,:,1),
matrizElementoEstruturante2(:,:,1),linhaOrigemElemento, colunaOrigemElemento ,
tipoProcessamento, tipoAdjuncao, tipoFuncao, tipoFuzzificacao);
   matrizResultante(:,:,2) = processamentoImagensAuxiliar( matrizImagem(:,:,2),
matrizElementoEstruturante2(:,:,2),linhaOrigemElemento, colunaOrigemElemento ,
tipoProcessamento, tipoAdjuncao, tipoFuncao, tipoFuzzificacao);
   matrizResultante(:,:,3) = processamentoImagensAuxiliar( matrizImagem(:,:,3),
matrizElementoEstruturante2(:,:,3),linhaOrigemElemento, colunaOrigemElemento ,
tipoProcessamento, tipoAdjuncao, tipoFuncao, tipoFuzzificacao);
   matrizResultante2(:,:,1) = desfuzzificacao(matrizResultante(:,:,1), tipoFuzzificacao);
   matrizResultante2(:,:,2) = desfuzzificacao(matrizResultante(:,:,2), tipoFuzzificacao);
   matrizResultante2(:,:,3) = desfuzzificacao(matrizResultante(:,:,3), tipoFuzzificacao);

   //nomeImagem = 'ImagemResultante.jpg';
   //imwrite (matrizResultante2, nomeImagem);
```

```
    //imagemResultante = imread(nomeImagem); // Lê a imagem resultante que foi salva no arquivo 'TipoDoProcessamento.jpg'
    //imshow(imagemResultante); // Mostra a imagem resultante na tela
endfunction

function [matrizResultante]=processamentoImagensAuxiliar(matrizImagem, matrizElementoEstruturante, linhaOrigemElemento, colunaOrigemElemento, tipoProcessamento, tipoAdjuncao, tipoFuncao, tipoFuzzificacao)

linhasMatriz = size(matrizImagem, 1); // inteiro que representa o numero de linhas da matriz da imagem
colunasMatriz = size(matrizImagem, 2);// inteiro que representa o numero de colunas da matriz da imagem

linhasElemento = size(matrizElementoEstruturante, 1); // inteiro que representa o numero de linhas da matriz do elemento estruturante
colunasElemento = size(matrizElementoEstruturante, 2); //i nteiro que representa o numero de colunas da matriz do elemento estruturante

// matrizImagemAuxiliar é uma matriz que contém os dados da imagem e uma borda com valores 256.
// O tamanho da borda depende das dimensões do elemento estruturante.
// O valor 256 foi escolhido para que a borda não interfira no valor do
// pixel resultante, pois nenhum pixel possui um valor maior do que 255.
matrizImagemAuxiliar = ones(linhasMatriz + linhasElemento, colunasMatriz + colunasElemento) * 510;

matrizImagemAuxiliar(linhaOrigemElemento:linhasMatriz + linhaOrigemElemento - 1,colunaOrigemElemento:colunasMatriz + colunaOrigemElemento - 1)=matrizImagem;

// matrizResultante é a matriz que irá armazenar os dados da imagem resultante do processamento
matrizResultante = zeros (linhasMatriz, colunasMatriz);

if (tipoProcessamento <3)
    // Erosao Godel:
    // E (imagem, elemento)(x) = inf (1 se elemento(x)<= imagem(x), imagem(x) se elemento(x) > imagem(x))
    //disp('escolhe')
    fuzzyElemento =fuzzificacao(double(matrizElementoEstruturante), tipoFuzzificacao); //Matriz do Elemento estruturante fuzzificada

    matrizImagemAuxiliarFuzzyficada = fuzzificacao(matrizImagemAuxiliar, tipoFuzzificacao); //Matriz da Imagem fuzzificada

    //aqui se pega os parametros para usar elemento estruturante circular
    [y , x , r]=size(matrizElementoEstruturante);
    r=y/2;

    //--aqui todo processamento Godel
    // Essa instrucao verifica se foi selecionado 'Erosao Godel função x'
    if (tipoProcessamento == 1 & tipoAdjuncao == 1 & tipoFuncao==1)
        //disp('godel')
```

```
      for m = 1: linhasMatriz
         //disp('for a')
         //disp(['coluna ' num2str(linhasMatriz) ' c  ' num2str(m)  ] );
         for n = 1 : colunasMatriz
            //disp('for b')
            fuzzyImagem = matrizImagemAuxiliarFuzzyficada(m : m + linhasElemento - 1, n : n +
colunasElemento - 1); //Matriz da Imagem fuzzificada

            valorFinal = 2; // O valorFinal é inicializado com 2 para que ele nunca seja o menor

            //Para cada pixel da imagem, esses laços percorrem o conjunto dos pixels que interferem no
valor do pixel em questão
            for l = 1: linhasElemento

               for c = 1: colunasElemento
                  if (fuzzyImagem(l,c) ~= 2  & fuzzyElemento(l,c) >= 0 & fuzzyElemento(l,c) <= 1) // Se o
pixel da imagem não for parte da borda da imagem auxiliar
                     if fuzzyElemento(l,c) <= fuzzyImagem(l,c) // Se elemento(x)<= imagem(x)
                        valorFinal = min(valorFinal, 1); // Escolhe o menor valor entre 1 e os outros valores
calculados
                     else // Se elemento(x) > imagem(x)
                        valorFinal = min(valorFinal, fuzzyImagem(l,c)); // Esolhe menor valor entre
                        // os outros valores calculados
                        // e o pixel da imagem
                     end
                  end
               end
            end
            matrizResultante(m,n) = valorFinal; // O pixel recebe o ínfimo dos valores calculados
anteriormente
         end

      end
   end
   // Essa instrucao verifica se foi selecionado 'Dilatacao Godel função x'
   if (tipoProcessamento == 2 & tipoAdjuncao == 1 & tipoFuncao==1) for m = 1: linhasMatriz
         //disp(['coluna ' num2str(linhasMatriz) ' c  ' num2str(m)  ] );
         for n = 1 : colunasMatriz

            fuzzyImagem = matrizImagemAuxiliarFuzzyficada(m : m + linhasElemento - 1, n : n +
colunasElemento - 1); //Matriz da Imagem fuzzificada

            valorFinal = 0; // O valorFinal é inicializado com 0 para que ele nunca seja o maior

            //Para cada pixel da imagem, esses laços percorrem o conjunto dos pixels que interferem no
valor do pixel em questão
            for l = 1: linhasElemento

               for c = 1: colunasElemento
```

```
                    if (fuzzyImagem(l,c) ~= 2  & fuzzyElemento(l,c) >= 0 & fuzzyElemento(l,c) <= 1) // Se o
pixel da imagem não for parte da borda da imagem auxiliar
                        valorFinal = max(valorFinal, min(fuzzyElemento(l,c),fuzzyImagem(l,c))); // Calcula
sup[min[B(x),g(x)]]
                    end
                end
            end
        matrizResultante(m,n) = valorFinal; // O pixel recebe o ínfimo dos valores calculados
anteriormente
        end
    end
end

//--aqui todo processamento Goguen
// Essa instrucao verifica se foi selecionado 'Erosao Goguen função x'
if (tipoProcessamento == 1 & tipoAdjuncao == 2 & tipoFuncao==1)
    for m = 1: linhasMatriz
        //disp(['coluna ' num2str(linhasMatriz) ' c  ' num2str(m)  ] );
        for n = 1 : colunasMatriz

            fuzzyImagem = matrizImagemAuxiliarFuzzyficada(m : m + linhasElemento - 1, n : n +
colunasElemento - 1); //Matriz da Imagem fuzzificada

            valorFinal = 2; // O valorFinal é inicializado com 2 para que ele nunca seja o menor

            //Para cada pixel da imagem, esses laços percorrem o conjunto dos pixels que interferem no
valor do pixel em questão
            for l = 1: linhasElemento

                for c = 1: colunasElemento
                    if (fuzzyImagem(l,c) ~= 2  & fuzzyElemento(l,c) >= 0 & fuzzyElemento(l,c) <= 1)  // Se o
pixel da imagem não for parte da borda da imagem auxiliar
                        if fuzzyElemento(l,c) <= fuzzyImagem(l,c) // Se elemento(x)<= imagem(x)
                            valorFinal = min(valorFinal, 1);  // Escolhe o menor valor entre 1 e os outros valores
calculados
                        else // Se elemento(x) > imagem(x)
                            valorFinal = min(valorFinal, fuzzyImagem(l,c)/fuzzyElemento(l,c)); // Esolhe menor
valor entre os outros
                            // valores calculados e a divisão do pixel
                            // da imagem pelo elemento
                        end
                    end
                end
            end
        matrizResultante(m,n) = valorFinal; // O pixel recebe o ínfimo dos valores calculados
anteriormente
        end
    end
end
// Essa instrucao verifica se foi selecionado 'Dilatacao Goguen função x'
```

```
   if (tipoProcessamento == 2 & tipoAdjuncao == 2 & tipoFuncao==1)
      for m = 1: linhasMatriz
         //disp(['coluna ' num2str(linhasMatriz) ' c  ' num2str(m)  ] );
         for n = 1 : colunasMatriz

            fuzzyImagem = matrizImagemAuxiliarFuzzyficada(m : m + linhasElemento - 1, n : n +
colunasElemento - 1); //Matriz da Imagem fuzzificada

            valorFinal = 0; // O valorFinal é inicializado com 2 para que ele nunca seja o maior

            //Para cada pixel da imagem, esses laços percorrem o conjunto dos pixels que interferem no
valor do pixel em questão
            for l = 1: linhasElemento

               for c = 1: colunasElemento
                  if (fuzzyImagem(l,c) ~= 2  & fuzzyElemento(l,c) >= 0 & fuzzyElemento(l,c) <= 1) // Se o
pixel da imagem não for parte da borda da imagem auxiliar
                     valorFinal = max(valorFinal, fuzzyImagem(l,c).*fuzzyElemento(l,c)); // Calcula
sup[B(x),g(x)]
                  end
               end
            end
      matrizResultante(m,n) = valorFinal; // O pixel recebe o ínfimo dos valores calculados
anteriormente
         end
      end
   end

   //--aqui todo processamento Lukasiewicz
   // Essa instrucao verifica se foi selecionado 'Erosao Lukasiewicz função x'
   if (tipoProcessamento == 1 & tipoAdjuncao == 3 & tipoFuncao==1)
      for m = 1: linhasMatriz
         //disp(['coluna ' num2str(linhasMatriz) ' c  ' num2str(m)  ] );
         for n = 1 : colunasMatriz

            fuzzyImagem = matrizImagemAuxiliarFuzzyficada(m : m + linhasElemento - 1, n : n +
colunasElemento - 1); //Matriz da Imagem fuzzificada

            valorFinal = 2; // O valorFinal é inicializado com 2 para que ele nunca seja o menor

            //Para cada pixel da imagem, esses laços percorrem o conjunto dos pixels que interferem no
valor do pixel em questão
            for l = 1: linhasElemento

               for c = 1: colunasElemento
                  if (fuzzyImagem(l,c) ~= 2  & fuzzyElemento(l,c) >= 0 & fuzzyElemento(l,c) <= 1) // Se o
pixel da imagem não for parte da borda da imagem auxiliar
                     valorFinal = min(valorFinal, min(1, 1 - fuzzyElemento(l,c)+fuzzyImagem(l,c))); //
Calcula min [1, lambda[B(x)] + lambda[1 - A(x)]]
                  end
```

```
                end
            end
        matrizResultante(m,n) = valorFinal; // O pixel recebe o ínfimo dos valores calculados
anteriormente
            end
        end
    end
    // Essa instrucao verifica se foi selecionado 'Dilatação Lukasiewicz função x'
    if (tipoProcessamento == 2 & tipoAdjuncao == 3 & tipoFuncao==1)
        for m = 1: linhasMatriz
            //disp(['coluna ' num2str(linhasMatriz) ' c  ' num2str(m)  ] );
            for n = 1 : colunasMatriz

                fuzzyImagem = matrizImagemAuxiliarFuzzyficada(m : m + linhasElemento - 1, n : n +
colunasElemento - 1); //Matriz da Imagem fuzzificada

                valorFinal = 0; // O valorFinal é inicializado com 2 para que ele nunca seja o maior

                //Para cada pixel da imagem, esses laços percorrem o conjunto dos pixels que interferem no
valor do pixel em questão
                for l = 1: linhasElemento

                    for c = 1: colunasElemento
                        if (fuzzyImagem(l,c) ~= 2  & fuzzyElemento(l,c) >= 0 & fuzzyElemento(l,c) <= 1) // Se o
pixel da imagem não for parte da borda da imagem auxiliar
                            valorFinal = max(valorFinal, max(0, fuzzyElemento(l,c)+fuzzyImagem(l,c) - 1)); //
Calcula max [0, lambda[B(x)] + lambda[A(x) - 1]]
                        end
                    end
                end
        matrizResultante(m,n) = valorFinal; // O pixel recebe o ínfimo dos valores calculados
anteriormente
            end
        end
    end
end

// Caso o usuário selecione uma abertura, primeiro a matriz da imagem é
// erodida e depois é dilatada
if (tipoProcessamento == 3)
    matrizErodida =  processamentoImagensAuxiliar(matrizImagem,
matrizElementoEstruturante,linhaOrigemElemento, colunaOrigemElemento , 1, tipoAdjuncao,
tipoFuncao, tipoFuzzificacao);
    matrizErodidaEdesfuzzificada = desfuzzificacao(matrizErodida, tipoFuzzificacao);
    matrizDilatada = processamentoImagensAuxiliar(matrizErodidaEdesfuzzificada,
matrizElementoEstruturante,linhaOrigemElemento, colunaOrigemElemento , 2, tipoAdjuncao,
tipoFuncao, tipoFuzzificacao);
    matrizResultante = matrizDilatada;
end
```

```
// Caso o usuário selecione um fechamento, primeiro a matriz da imagem é
// dilatada e depois é erodida
if (tipoProcessamento == 4)
   matrizDilatada =  processamentoImagensAuxiliar(matrizImagem,
matrizElementoEstruturante,linhaOrigemElemento, colunaOrigemElemento , 2, tipoAdjuncao,
tipoFuncao, tipoFuzzificacao);
   matrizDilatadadesfuzzificada = desfuzzificacao(matrizDilatada, tipoFuzzificacao);
   matrizErodida = processamentoImagensAuxiliar(matrizDilatadadesfuzzificada,
matrizElementoEstruturante,linhaOrigemElemento, colunaOrigemElemento , 1, tipoAdjuncao,
tipoFuncao, tipoFuzzificacao);
   matrizResultante = matrizErodida;
end
endfunction

// Função que retorna os valores dos pixels (0 a 255) em uma escala de 0 a 1
function [matrizNormalizada]=fuzzificacao(matriz, tipoNormalizacao)

if tipoNormalizacao == 1 //Se a funçao x/255 for escolhida
   matrizNormalizada = double(matriz./255);
end

if tipoNormalizacao == 2 // Se a funçao seno for escolhida
   matrizNormalizada = double(sin((matriz/255)*(pi/2)));
end

if tipoNormalizacao == 3 // Se a funçao tangente for escolhida
   matrizNormalizada = double(((tan((((matriz/255)*90+135)/360) * 2*pi))-1+2)/2);
end

endfunction

// Função que retorna os valores dos pixels (0 a 255) em uma escala de 0 a 255
function [matrizDesfuzzificada]=desfuzzificacao(matriz, tipoFuzzificacao)

if tipoFuzzificacao == 1
   matrizDesfuzzificada = (matriz*255);
end

if tipoFuzzificacao == 2 // Se a fuzzificação seno for escolhida
   matrizDesfuzzificada = ((asin(matriz)/(pi/2)) * 255);
end

if tipoFuzzificacao == 3 // Se a fuzzificação tangente for escolhida
   matrizDesfuzzificada = ((((atan((matriz*2)-1))/(2*pi))*360+45)/90 * 255);
end

endfunction
```