DISSERTAÇÃO DE MESTRADO

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM

CIÊNCIA DA COMPUTAÇÃO

"Otimização Inteligente da Decomposição de Elemento Estruturante Em Morfologia Matemática Binária"

ALUNO: Paulo Eduardo Pillon

ORIENTADOR: Prof. Dr. Emerson Carlos Pedrino

São Carlos
Janeiro/2014

CAIXA POSTAL 676
FONE/FAX: (16) 3351-8233
13565-905 - SÃO CARLOS - SP
BRASIL

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

OTIMIZAÇÃO INTELIGENTE DA DECOMPOSIÇÃO DE ELEMENTO ESTRUTURANTE EM MORFOLOGIA MATEMÁTICA BINÁRIA

PAULO EDUARDO PILLON

ORIENTADOR: PROF. DR. EMERSON CARLOS PEDRINO

São Carlos - SP
Novembro/2014

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

OTIMIZAÇÃO INTELIGENTE DA DECOMPOSIÇÃO DE ELEMENTO ESTRUTURANTE EM MORFOLOGIA MATEMÁTICA BINÁRIA

PAULO EDUARDO PILLON

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: PIS.
Orientador: Prof. Dr. Emerson Carlos Pedrino

São Carlos - SP
Novembro/2014

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

P642oi

Pillon, Paulo Eduardo.
Otimização inteligente da decomposição de elemento estruturante em morfologia matemática binária / Paulo Eduardo Pillon. -- São Carlos : UFSCar, 2015.
92 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2014.

1. Processamento de imagens. 2. Morfologia matemática. 3. Algoritmos genéticos. 4. Decomposição do elemento estruturante. I. Título.

CDD: 006.42 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Otimização Inteligente de Elemento Estruturante em Morfologia Matemática Binária"

Paulo Eduardo Pillon

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Emerson Carlos Pedrino
(Orientador - DC/UFSCar)

Prof. Dr. Valentin Obac Roda
(UFRN)

Prof. Dr. Mário Luiz Tronco
(EESC/USP)

São Carlos
Março/2014

DEDICATÓRIA

Dedico este trabalho de mestrado ao meu pai Paulo.

AGRADECIMENTOS

Agradeço a Deus por tudo que me proporcionou durante esses quase três anos de estudos: pela minha família, meus amigos e colegas de trabalho. Todos em conjunto colaboraram para o meu crescimento intelectual durante todo o percurso acadêmico, desde o início até a conclusão. Agradeço ao Ricardo pelas dispensas e compreensões em todas as vezes que precisei ir até a UFSCAR. Agradeço ao meu orientador professor Emerson pelas ideias e momentos de aprendizado. Agradeço aos professores Saulo e Cristina da UNICEP pelo apoio e incentivo em estar neste programa de pós graduação. Agradeço aos professores presentes na minha banca de qualificação. Agradeço em especial ao Paulinho por todos os momentos vividos, sempre me apoiando e incentivando a nunca desistir.

RESUMO

Ao trabalhar com imagens binárias, a morfologia matemática oferece ferramentas eficientes para análise das mesmas. Juntamente com algoritmos genéticos, permitem que sejam executadas de forma autônoma diversas tarefas computacionais. Nas últimas décadas tem-se utilizado a decomposição dos elementos estruturantes envolvidos nas operações morfológicas para diminuir o grau de dificuldade imposto por imagens de tamanho elevado implicando na eficiência do *hardware* utilizado para o processamento de tais imagens. Para acelerar esse processo, este trabalho de mestrado, utiliza os *softwares* MATLAB e Visual Studio como ferramentas de desenvolvimento, tendo como objetivo elaborar um algoritmo eficiente de decomposição do elemento estruturante facilitando uma possível aplicação em *hardware*, diminuindo ainda mais o tempo gasto no processamento de imagens.

Palavras-chave: Morfologia Matemática, Algoritmos Genéticos, Decomposição do Elemento Estruturante.

ABSTRACT

When working with binary images, mathematical morphology provides powerful tools for analyzing them. Along with genetic algorithms, it allows various computational tasks that are performed autonomously. In recent decades, the decomposition of structuring elements has been involved in the morphological operations used to reduce the degree of difficulty imposed by images of larger size implying in the efficiency of the hardware used for the processing of such images. For accelerating this process, this master's project uses MATLAB and Visual Studio as development tools, aiming at developing an efficient algorithm for decomposing the structuring element facilitating a possible hardware implementation with aim of decreasing the time waste in processing those images.

Keywords: Mathematical Morphology, Genetic Algorithms, Decomposition of Structuring Element.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG - *Algoritmo Genético*

FPGA - *Field-Programmable Gate Array*

MATLAB - *MATrix LABoratory*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Neste capítulo é exposto o contexto no qual este trabalho está inserido e a motivação que deu origem a esta pesquisa. Também são discutidos os objetivos a serem alcançados e demonstrada a organização desta dissertação.

1.1 Contexto

Este trabalho consistiu em testar e avaliar diversos procedimentos computacionais para a decomposição eficiente dos elementos estruturantes para morfologia matemática binária visando uma simplificação do processo elaborado por Shih (SHIH; WU, 2005) descrito em seu artigo.

Para este algoritmo utilizam-se técnicas de processamento digital de imagens e computação evolutiva facilitando o entendimento para futura implementação em *hardware* reconfigurável.

1.2 Motivação e Objetivos

Ao trabalhar com processamento morfológico de imagens, são encontradas muitas dificuldades, principalmente quando o sistema projetado não suporta imagens maiores que um determinado tamanho (quantidade de *pixels*). Para isso a decomposição dessas imagens em elementos menores juntamente com a utilização

de algoritmos genéticos é de extrema importância para o processamento das mesmas.

Com base nos parâmetros fornecidos por Shih (SHIH; WU, 2005) em seu trabalho, determinando que o comprimento dos cromossomos seja variado, que a imagem original não seja dividida (tamanho real) e que a quantidade de operadores morfológicos (dilatação e união) devem variar dependendo do tipo de imagem, torna-se inviável a implementação em *hardware* onde muitas vezes este é limitado. Tendo em vista que o objetivo deste trabalho é a simplificação da decomposição proposta por Shih, padronizaram-se o comprimento do cromossomo e a quantidade das operações executadas nos elementos estruturantes, tornando o uso de memória fixo para alocação dos elementos e execução das operações de dilatação e união. Isto acelera o processo de tratamento da imagem original elaborando uma estrutura simples que permite a implementação em *hardware* utilizando FPGA e programação paralela, trazendo ainda mais rapidez para a execução do algoritmo proposto para a decomposição.

1.3 Organização da Dissertação

Esta dissertação de mestrado está estruturada em cinco capítulos, onde são apresentados:

- O Capítulo 1, denominado introdução, que destaca os principais objetivos desta pesquisa detalhando o contexto ao qual está inserido, e os motivos que levaram a elaboração do trabalho.
- O Capítulo 2, onde se aborda a base teórica necessária para dar consistência às técnicas utilizadas no tema proposto a partir da literatura pesquisada.
- O Capítulo 3, composto pela metodologia empregada no desenvolvimento do trabalho, além das técnicas para implementação do algoritmo desenvolvido.
- O Capítulo 4, onde se discutem os resultados obtidos em toda a pesquisa.
- Por fim, a Conclusão destaca a contribuição das ideias abordadas e possíveis trabalhos futuros.

Capítulo 2

REVISÃO BIBLIOGRÁFICA

Neste capítulo encontram-se os conceitos relacionados à elaboração do trabalho de mestrado. São apresentadas as considerações iniciais (seção 2.1), conceitos sobre Morfologia Matemática (seção 2.2), Algoritmos Genéticos (seção 2.3), Métodos de Decomposição do Elemento Estruturante (seção 2.4) e por fim, apresentam-se as considerações finais (seção 2.5).

2.1 Considerações Iniciais

A utilização dos operadores básicos da morfologia matemática aliados aos algoritmos genéticos (baseados em processos naturais de evolução) e as técnicas de decomposição de elementos estruturantes permitem solucionar diversos problemas de processamento de imagens (em específico neste trabalho, no tratamento de imagens binárias), ao qual está inserido o contexto deste trabalho. Assim, neste capítulo é descrita a fundamentação teórica desenvolvida a partir da pesquisa dos domínios citados acima.

2.2 Morfologia Matemática

A palavra *morfologia*, segundo Gonzalez e Woods (GONZALEZ; WOODS, 2002), é empregada na biologia para caracterizar o estudo da forma e da estrutura dos seres vivos. Contudo, esta palavra também é usada no contexto matemático,

por meio da teoria de conjuntos, como ferramenta para extração de componentes de imagens para a representação e descrição de uma região. Usando a teoria dos conjuntos como linguagem, a morfologia matemática oferece uma abordagem para vários problemas de processamento de imagens, visando representar as formas dos objetos de uma imagem. Em imagens binárias, que é o foco deste trabalho, o conjunto em questão é o dos números inteiros $\mathbb{Z}^2$, onde cada elemento é um vetor bidimensional de coordenadas (x, y) dos *pixels* ativos da imagem (podendo ser brancos ou pretos, Figura 2.1).

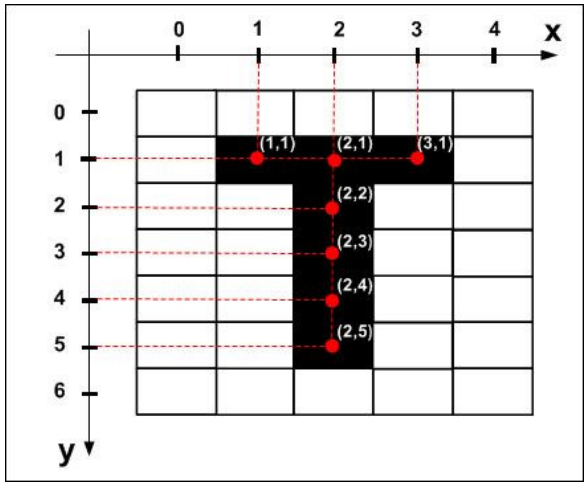

Figura 2.1 - Descrição de uma imagem binária em formato de "T".

Segundo Facon (FACON, 1996), com o auxílio da teoria dos conjuntos pode-se extrair de uma imagem muitas informações, como: segmentação, realce, detecção de bordas, dentre outras. Para isso, é necessária a utilização de um "elemento estruturante" (um conjunto com forma e tamanho definidos, Figura 2.2) e sua escolha está diretamente relacionada ao resultado da operação morfológica.

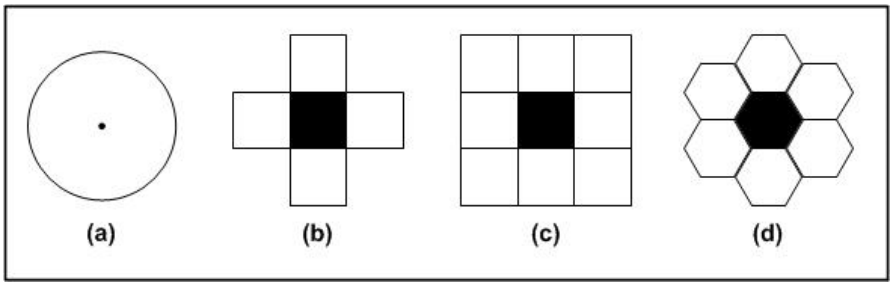

Figura 2.2 - Elementos estruturantes básicos
(a) Disco (b) Cruz (c) Quadrado (d) Hexágono. Fonte: Adaptado de (SOILLE, 1999).

2.2.1 Combinação de Conjuntos: União e Intersecção

Segundo Wangenheim (WANGENHEIM, 1998), A união de dois conjuntos A e B é o conjunto de todos os elementos que pertencem a A ou a B ou a ambos, dado por:

$$\boldsymbol{A \cup B = \{a |\, a \in A \text{ ou } a \in B\}} \quad \textbf{(2.1)}$$

A interseção de dois conjuntos A e B é o conjunto de todos os elementos que pertencem a A e a B dado por (WANGENHEIM, 1998):

$$\boldsymbol{A \cap B = \{a |\, a \in A \; e \; a \in B\}} \quad \textbf{(2.2)}$$

2.2.2 Operações Básicas em Imagens Binárias: Dilatação e Erosão

Sejam A e B conjuntos de $\mathbb{Z}^2$e $\emptyset$ um conjunto vazio, define-se a dilatação de A (imagem original) por B (elemento estruturante), denotada por $A \oplus B$, como (GONZALEZ; WOODS, 1992):

$$\boldsymbol{A \oplus B = \{x | \left(\widehat{B}\right)_x \cap A \neq \emptyset\}} \quad \textbf{(2.3)}$$

Onde $(\widehat{\boldsymbol{B}})_{\boldsymbol{x}}$ representa a translação da reflexão de $\boldsymbol{B}$. A dilatação de A por B é então, o conjunto de todos os deslocamentos x tais que a reflexão de B e A se sobreponham em pelo menos um elemento não nulo (PEDRINO, 2008).

O conjunto $\boldsymbol{B}$, também chamado de *elemento estruturante*, tem a função de percorrer a imagem e verificar a possibilidade da existência de uma intersecção com os vizinhos mais próximos ao *pixel* x. Se a resposta satisfazer a condição imposta, a posição central dentro da imagem resultante recebe o valor "1", sendo falsa, o valor "0".

A dilatação também pode ser obtida utilizando adição vetorial dado por (GONZALEZ & WOODS, 1992; PARKER, 1997; RUSS, 1995; MARQUES & VIEIRA, 1999):

$$\boldsymbol{A} \oplus \boldsymbol{B} = \{\boldsymbol{c} | \boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}, \boldsymbol{a} \in \boldsymbol{A}, \boldsymbol{b} \in \boldsymbol{B}\} \quad \textbf{(2.4)}$$

Onde $\boldsymbol{A}$ representa o conjunto a ser operado e $\boldsymbol{B}$ é um segundo conjunto denominado de elemento estruturante, o qual sua composição define a natureza específica da dilatação. Sendo assim, $\boldsymbol{c}$ é o resultado da adição de todos os vetores $\boldsymbol{a}$ e $\boldsymbol{b}$.

Na Figura 2.4 são apresentadas duas imagens binárias, uma original (2.4.a) e a outra dilatada (2.4.b) pela aplicação do operador morfológico de dilatação através do elemento estruturante da Figura 2.3.

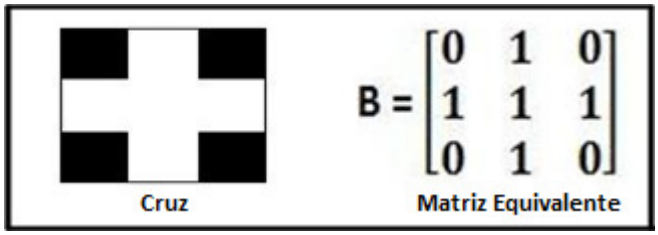

Figura 2.3 - Ilustração de um elemento estruturante.

É possível perceber que esse processo tem a capacidade de aumentar os objetos da imagem, preenchendo possíveis lacunas e consequentemente unir objetos.

Figura 2.4 - Aplicação do operador de dilatação
(a) Imagem Original. (b) Imagem após a dilatação.

Sejam A e B conjuntos de $\mathbb{Z}^2$, assim, a erosão de A por B, denotada por$A \ominus B$, é dada por (GONZALEZ; WOODS, 1992):

$$\boldsymbol{A \ominus B = \{x|(B)_x \subseteq A\}} \quad \textbf{(2.5)}$$

Deste modo, a erosão de A por B é o conjunto de todos os pontos x, tais que B, quando transladado de x, fique contido em A (PEDRINO, 2008). Isto é, o elemento estruturante B deve percorrer o conjunto A, fazendo-se uma comparação entre os *pixels* vizinhos de x. Se nessa comparação todos os *pixels* forem idênticos, o *pixel* de saída fica ativo, caso contrário, o *pixel* fica inativo.

A erosão também pode ser obtida usando vetores de subtração dado por (GONZALEZ & WOODS, 1992; PARKER, 1997; RUSS, 1995; MARQUES & VIEIRA, 1999):

$$\boldsymbol{A \ominus B = \{c|c + b \in A \text{ para todo } b \in B\}} \quad \textbf{(2.6)}$$

Na Figura 2.5 a aplicação do operador utilizando elemento estruturante da Figura 2.3 permite visualizar algumas das características básicas da erosão que se resumem em diminuir objetos, eliminar objetos menores que o elemento estruturante utilizado e aumentar lacunas, permitindo-se assim a separação de objetos próximos.

(a) (b)

Figura 2.5 - Aplicação do operador de erosão
(a) Imagem Original. (b) Imagem após erosão.

2.3 Algoritmos Genéticos

Na natureza, a existência de indivíduos com diferentes habilidades de sobreviver ao meio ambiente, está associada a variações de seus próprios cromossomos. Essas variações estão relacionadas tanto à estrutura como ao comportamento desses indivíduos em seu meu ambiente. Indivíduos mais aptos (com maior capacidade de executar tarefas) sobrevivem a uma taxa muito superior em relação aos menos aptos, conceito este descrito por Charles Darwin em seu livro sobre “A Origem das Espécies por Meio da Seleção Natural” do ano de 1859 (KOZA, 1992).

Baseado nesse processo natural de evolução surgiu a computação evolucionária, tornando-se uma ferramenta muito importante utilizando modelos computacionais para resolver problemas complexos. Apesar da infinidade de modelos existentes, todos têm em comum a simulação da evolução das espécies através de seleção, reprodução e mutação, os quais dependem do “desempenho” desses indivíduos dentro de um “ambiente” (LINDEN, 2008).

Os algoritmos genéticos, um ramo da computação evolucionária, são constituídos de uma população de estruturas (denominadas de indivíduos ou cromossomos), semelhantemente à evolução das espécies. Cada integrante dessa população recebe uma avaliação (*fitness*) que é calculada para qualificar o mesmo dentro da própria população indicando o “quão bom” ele é para a solução do problema (KOZA, 1992). Após essa avaliação são aplicados os operadores genéticos (recombinação e mutação, entre outros) de forma a simular a sobrevivência do mais apto.

O comportamento padrão desses algoritmos evolucionários se resume no seguinte pseudocódigo:

```
inicio
        inicializar(n)
        t <- 0 % Contador de tempo.
        função Inicializa_População_P(n) % Inicia população aleatoriamente de acordo valor n.
        enquanto não terminar faça % Condição de término.
                função Avalie_População_P(t) % Avaliação.
                P' <- função Selecione_Pais_P(t) % Seleção para gerar nova população.
                P' <- função Recombinação_e_mutação_P' % Operadores de reprodução.
                função Avalie_População_P' % Avaliação da nova população.
                P(t+1) <- Selecione_sobreviventes_P(t), P' % Seleciona sobreviventes.
                t <- t + 1 % Incrementa contador.
        fim enquanto
fim
```

Figura 2.6 - Pseudocódigo

Fonte: Adaptado de (LINDEN, 2008).

Este algoritmo consiste da geração de uma população inicial aleatória e avaliação de cada indivíduo seguido da seleção dos pais que servirão para gerar os indivíduos da nova população através de recombinações e/ou mutações. Essa nova população é avaliada e são selecionados os sobreviventes para a próxima geração. O *loop* é repetido diversas vezes (constituindo as gerações) até que seja alcançado o critério imposto (tempo, *fitness*, etc).

A codificação da informação juntamente da função de avaliação forma um ponto crucial dentro do Algoritmo Genético (AG). Assim como na natureza, a informação deve ser codificada em cromossomos e a reprodução (equivalente à reprodução sexuada, ou seja, uma combinação de pedaços entre dois genitores, garantindo a diversidade e gerando filhos mais aptos evoluindo a população ao passar das gerações, ao contrário da reprodução assexuada onde os filhos são idênticos aos pais, estando sujeitos apenas a variações causadas pela mutação) se encarregará da evolução da população (MITCHELL, 1996). A mutação cria diversidade, alterando os genes de forma menos frequente que no cruzamento. A função de avaliação deve ser executada de tal forma que os indivíduos mais aptos sejam selecionados com mais frequência que os menos aptos, fazendo com que as boas características permaneçam dentro da nova população. Este pode ser um ponto muito perigoso dentro do AG, pois se a seleção descartar de forma errada os menos aptos causaria uma rápida convergência genética para um conjunto semelhante de características e uma população de baixa diversidade genética, não

conseguindo evoluir, a não ser pela mutação que pode acarretar na não solução do problema.

2.3.1 Escolha da população inicial

Seguindo a ideia de que os algoritmos genéticos devem simular o processo de evolução existente no meio ambiente, a escolha da população inicial é o primeiro passo a ser dado e deve ser feito da maneira mais simples possível, tornando-se um processo aleatório independente para cada indivíduo. Como a população possui tamanho finito, não se pode garantir que a lei das probabilidades alcance todo o espaço de soluções, pois isso só ocorre para um número infinito de sorteios. A inicialização de modo aleatório gera uma melhor distribuição das soluções garantindo uma boa exploração sobre o espaço de busca (MITCHELL, 1996).

2.3.2 Função de avaliação (*fitness*)

Para determinar se um indivíduo será uma possível solução do problema em questão, ou seja, se o indivíduo é mais apto, usa-se a função de avaliação. Dada a generalidade dos AG's, a função de avaliação tem a função de deixar o problema mais perto da realidade, sendo possível o uso do mesmo programa para diversos casos, apenas trocando a função de avaliação por outra que se encaixe para satisfazer o novo problema.

Na escolha da função de avaliação deve-se observar que ela terá que conter tanto as restrições quanto os objetivos da qualidade, tendo assim o maior conhecimento possível sobre o problema. Outro ponto chave é que não é permitida uma avaliação nula ou negativa, fazendo-se assim com que a soma das avaliações diminua, alterando-se a seleção (este assunto será tratado na próxima seção), ou que haja mais de um elemento para o mesmo valor de avaliação. Além disso, se torna difícil a associação de um espaço na seleção para um número negativo. A ideia da avaliação é utilizar a técnica de maximização para uma função $f(x)$ qualquer, e caso seja necessário fazer com que se utilize o processo de minimização inverte-se a função de avaliação resultando em $f'(x) = 1/f(x)$. Para evitar que a função retorne um valor nulo, como citado acima, acrescenta-se uma constante ct

qualquer, transformando-se ambas as funções em $f(x) = f(x) + ct$ e $f'(x) = 1/f(x) + ct$.

2.3.3 Seleção dos pais

Como citado na seção anterior, para fazer a seleção dos indivíduos mais aptos para serem os pais, um dos métodos utilizados é o método da roleta (GOLDBERG, 1989). A seleção consiste em fornecer maior privilégio para os indivíduos que possuem avaliação mais alta, sem eliminar os mais baixos, pois para que seja alcançado o objetivo, todos os indivíduos podem conter características favoráveis a criação de um "superindivíduo".

Nesse método, é gerada uma roleta virtual onde cada cromossomo recebe uma fatia proporcional à sua avaliação (com isso o somatório dessas fatias não pode ultrapassar 100%). A partir deste ponto, a roleta é girada e o indivíduo selecionado será o que a roleta "parar sobre ele". Entretanto, no computador, não se pode "girar uma roleta", por isso o sorteio utiliza um algoritmo capaz de simular a roleta, como pode ser visto a seguir:

```
% Some todas as avaliações para uma variável soma
% Ordene todos os indivíduos em ordem crescente de avaliação (opcional)
% Selecione um número s satisfazendo a expressão 0 < s < soma
inicio
        inicializa(s)
        i <- 1 % contador de indivíduos
        aux <- avaliação do indivíduo (i) % aux recebe a soma das avaliações
        enquanto aux < s
                i <- i + 1
                aux <- aux + avaliação do indivíduo (i)
        fim enquanto
fim
```

Figura 2.7 - Exemplo roleta.
Fonte: Adaptado de (LINDEN, 2008).

O algoritmo tem parada garantida, visto que aux tende ao somatório das avaliações dos indivíduos e s é menor que esta soma e o índice do indivíduo é dado pela variável i.

A roleta pode ser montada na forma de uma Tabela para que o processo de sorteio do indivíduo seja melhor visualizado. Na Tabela 2.1, por exemplo, o indivíduo “0110” recebe um pedaço igual a 58,07% da roleta. Em contra partida, o indivíduo “0001” recebe uma fração inferior (1,6% da roleta) tornando suas chances de seleção bem menores, porém não são descartados como dito anteriormente.

Tabela 2.1 - Exemplo de Roleta em forma de tabela
Fonte: Adaptado de (LINDEN, 2008).

INDIVÍDUO	AVALIAÇÃO	PEDAÇO DA ROLETA (%)
0001	1	1,61
0011	9	14,51
0100	16	25,81
0110	36	58,07
TOTAL	62	100,00

2.3.4 Reprodução e seus operadores

Dentre os operadores utilizados em AG para reprodução, existem os operadores de cruzamento e o de mutação, tratados a seguir.

2.3.4.1 Cruzamento

O cruzamento, assim como na biologia, é a recombinação dos alelos do par de cromossomos selecionado chamados de cromossomos pais (HOLLAND, 1992). A operação que o cruzamento executa nos pais selecionados é extremamente simples, e depende de uma probabilidade para ocorrer. Em um dos métodos existentes, é selecionado um ponto de corte (posição entre dois genes em um cromossomo) sendo que um indivíduo de n genes contém n-1 pontos de corte possíveis. Este ponto define a separação do material genético dos pais, portanto, uma metade irá para um filho e a outra para o outro filho. Na Figura 2.11, o cromossomo utilizado como exemplo possui 5 genes e consequentemente 4 pontos possíveis de corte.

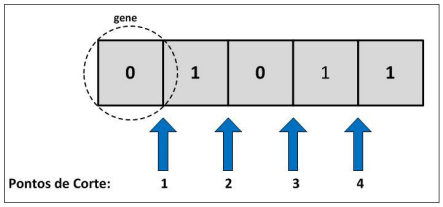

Figura 2.8 - Exemplo da seleção do ponto de corte na aplicação do operador de cruzamento. Fonte: Adaptado de (LINDEN, 2008).

Após o sorteio do ponto de corte onde, o cromossomo é divido em duas partes (Figura 2.8) que não precisam ser de mesmo tamanho, o primeiro filho recebe a concatenação da parte esquerda do primeiro pai com a parte direita do segundo pai e o segundo filho a concatenação das outras duas metades que restaram (direita e esquerda).

2.3.4.2 Mutação

O operador de mutação tem, por sua vez, uma probabilidade bem menor de ocorrência sobre o cromossomo do que o operador de cruzamento. Holland (HOLLAND, 1992), diz que sua aplicação consiste em fazer o sorteio em cada alelo dos genes do cromossomo e, caso seja atingida a probabilidade adotada, o valor deste gene é alterado (Figura 2.9).

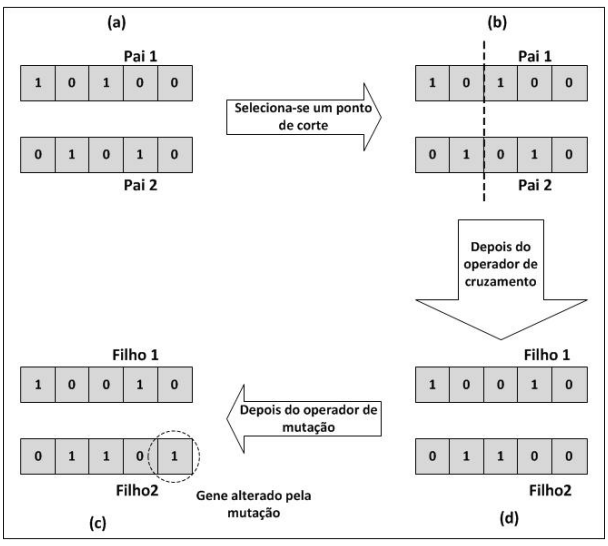

Figura 2.9 - Exemplo de aplicação dos operadores de cruzamento e mutação (a) Pais selecionados. (b) Ponto de corte. (c) Filhos gerados após o cruzamento. (d) Gene sofrendo mutação. Fonte: Adaptado de (LINDEN, 2008).

2.3.4.3 Taxa de Mutação e Cruzamento

Nos algoritmos genéticos clássicos, os operadores genéticos envolvidos, tais como cruzamento e mutação, trabalham prioritariamente com probabilidades constantes. Diferentes taxas de cruzamento e mutação podem, no entanto, atravessar diferentes direções de busca no espaço de estados, afetando o desempenho do algoritmo genético aplicado. Na verdade, o desempenho global de um algoritmo genético depende da manutenção de um nível aceitável de produtividade durante todo o processo de evolução. É, portanto, essencial para a concepção de um algoritmo genético a adaptação das taxas de cruzamento e mutação de forma apropriada. Segundo Wen-Yang (WEN-YAN et al., 2003), é importante avaliar se a probabilidade aplicada do operador está de acordo com a sua "contribuição" (ou produtividade), ou seja, se a taxa “escolhida” é potencialmente capaz de produzir filhos com melhora na sua aptidão.

2.3.5 Módulo de População

Depois da seleção dos pais e de sua respectiva reprodução feita pelos operadores anteriormente citados, cruzamento e mutação, o módulo de população tem como função substituir os pais menos aptos conforme os filhos são gerados, ou seja, caso os filhos sejam mais aptos que os pais que os geraram, os mesmos são promovidos à nova geração, caso contrário são descartados.

2.3.6 Critério de parada

Depois que todo o processo em volta da população inicial termina (*fitness*, seleção dos pais e reprodução), o algoritmo passa por uma avaliação denominada como “critério de parada”. Neste ponto é analisado se o algoritmo alcançou o critério imposto no início do problema para saber se foi encontrada uma solução ou não. Caso a resposta seja positiva, o algoritmo termina sua execução e retorna a resposta encontrada. Caso seja negativa, o algoritmo continua sua execução a partir do cálculo do *fitness* seguindo adiante até este mesmo ponto, caso permaneça com a resposta negativa, este ciclo (Figura 2.10) é repetido novamente até que seja encontrada uma solução para o problema.

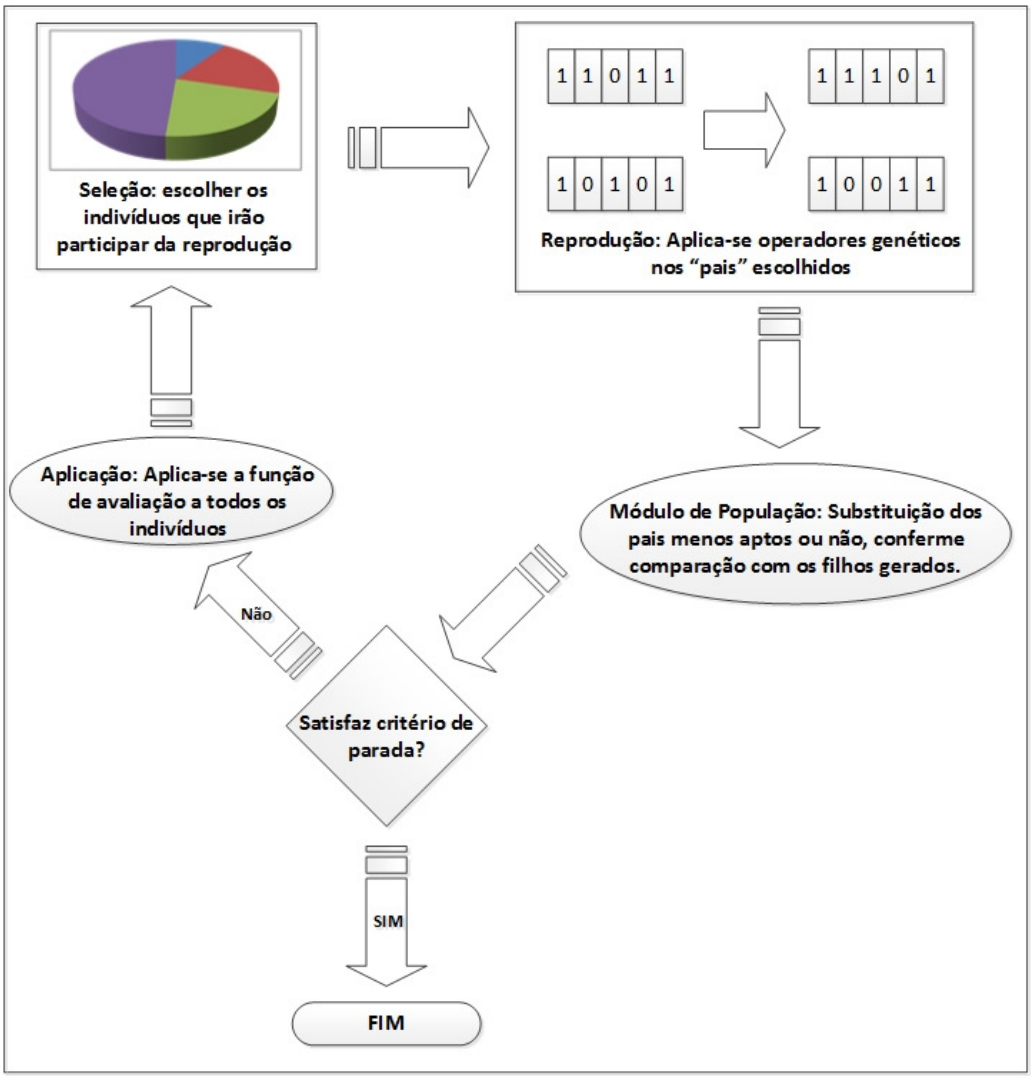

Figura 2.10 - Ciclo de execução do algoritmo genético.
Fonte: Adaptado de (LINDEN, 2008).

2.4 Métodos de Decomposição do Elemento Estruturante

A decomposição de um elemento estruturante é uma questão muito importante em sistemas de processamento de imagens morfológicas, principalmente quando tais sistemas não podem trabalhar com um tamanho muito grande de imagens, sendo esta, a única maneira de ultrapassar tal dificuldade. A ideia se baseia nas operações básicas da morfologia matemática para decompor um elemento estruturante qualquer em uma sequência de elementos estruturantes menores compostos por matrizes de tamanho 3x3 (ou seja, três linhas por três colunas) tornando possível a recursividade para se obter a imagem original.

Segundo Zhuang e Halarick (ZHUANG; HALARICK, 1986), um elemento estruturante $\boldsymbol{S}$, convexo e de tamanho finito, é decomposto em $\boldsymbol{N}$ elementos estruturantes $\boldsymbol{H_1, \ldots, H_N}$, se e somente se:

$$\boldsymbol{S = H_1 \oplus \ldots \oplus H_N} \quad \textbf{(2.7)}$$

O problema nesse caso passa a ser a busca pelo menor $\boldsymbol{N}$, se existir, determinado previamente pela limitação de *hardware*. E para determinar a decomposição de $\boldsymbol{S}$, se existir uma, é necessário um processo de busca combinatória. Zhuang e Halarick (ZHUANG; HALARICK, 1986) sugeriu que a busca fosse feita pelo método da árvore (Figura 2.11), começando pela raiz, de tamanho $\boldsymbol{m}$ de todos os elementos estruturantes possíveis.

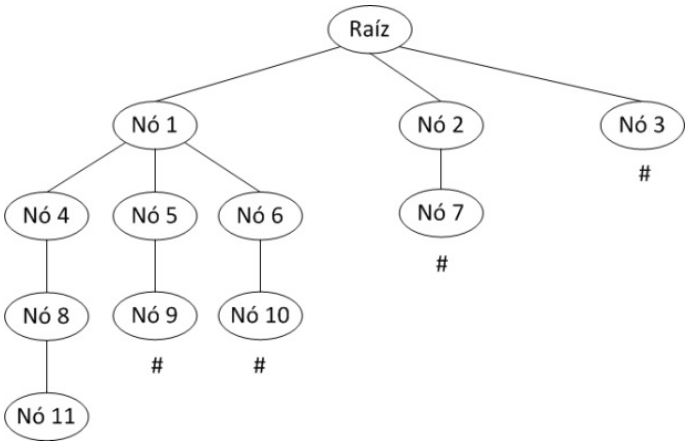

Figura 2.11 - Busca em árvore
Fonte: Adaptado de (ZHUANG; HALARICK, 1986).

A partir desse ponto, começa-se uma busca em todos os nós filhos eliminando aqueles que não farão parte da solução. Se não existir um nó sobrevivente, a busca termina e $\boldsymbol{S}$ não pode ser decomposto. Sendo $\boldsymbol{H_i}$ uma matriz 3x3 (equação 2.11), $\boldsymbol{S}$ passa a ser um conjunto de dilatações de elementos menores e tal decomposição torna possível a dilatação de uma imagem $\boldsymbol{X}$ por $\boldsymbol{S}$ usando este elemento 3x3 de uma maneira recursiva, dada por (PARK; CHIN, 1995):

$$\boldsymbol{X \oplus S = ((X \oplus H_1) \oplus H_2) \oplus H_N} \quad \textbf{(2.8)}$$

Uma estratégia alternativa para a decomposição de $\boldsymbol{S}$ seria minimizar o número de *pixels* ativos em cada matriz $\boldsymbol{H_i}$ necessário para a representação deste conjunto. Em contrapartida, isso maximiza o número de matrizes (ou seja, $\boldsymbol{N}$) que por sua vez segue em sentido oposto ao que está sendo procurado (menor $\boldsymbol{N}$) (RICHARDSON; SCHAFER, 1991).

A dilatação mostrada na equação 2.11, de acordo com Park (PARK; CHIN, 1994), pode ser visualizada pela decomposição de imagens convexas em treze "fatores primos" de elementos estruturantes de tamanho 3x3, como é mostrado na Figura 2.12. Xu (XU, 1991) diz que esse conjunto de treze fatores tem a propriedade de serem todos convexos e indecompostos e de não serem triviais (ou não singulares, isto é, que o conjunto não possui somente um elemento).

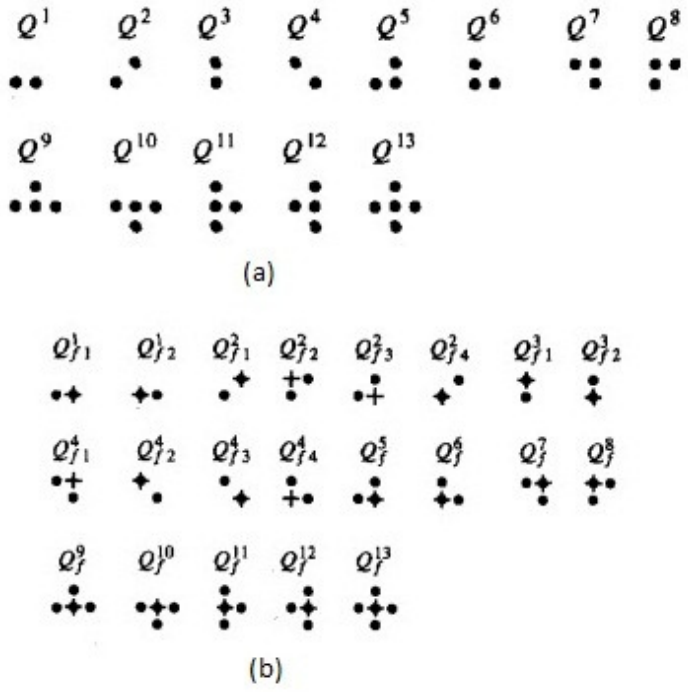

Figura 2.12 - Treze fatores primos para imagens convexas
(a**) sem origem definida (b) com origem definida**
Fonte: (PARK; CHIN, 1994).

Em meados da década de 90, Park (PARK; CHIN, 1995) utilizando um método criado por Freeman (FREEMAN, 1974), chamado de "código da cadeia", propôs a decomposição de formas arbitrárias de elementos estruturantes pois, até o momento, só haviam sido feitas pesquisas a respeito de imagens convexas. O "Código da cadeia" é uma técnica de codificação para imagens, ou seja,

basicamente é um vetor composto por números entre 0 e 3 ou 0 e 7 (Figura 2.13), tomando como base inicial um *pixel* qualquer situado na borda da imagem, analisando *pixel* a *pixel* qual a direção do seu próximo vizinho (utilizando 4-vizinhos ou 8-vizinhos) em todo o contorno da imagem (o sentido pode ser horário ou anti-horário dependo da convenção adotada).

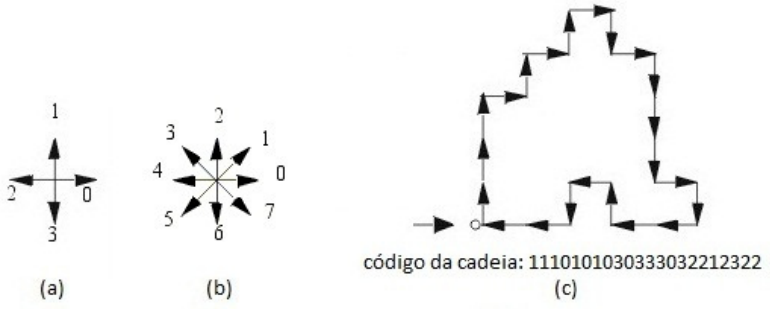

Figura 2.13 - Exemplo de código da cadeia
(a) 4-vizinhos (b) 8-vizinhos (c) código da cadeia gerado para o caso (a)
Fonte: Adaptado de (FREEMAN, 1974).

Existem imagens que são impossíveis de serem decompostas, para isso deve ser feita uma análise onde, eliminando essas imagens, tem-se um conjunto menor composto tanto por imagens que podem ser decompostas como por imagens que não podem ser (porém em número reduzido). Com esse conjunto menor e fazendo outra análise, forma-se o grupo onde se encontram todas as imagens que podem ser decompostas que inclui as imagens côncavas e as imagens convexas que é um subconjunto desse conjunto (Figura 2.14).

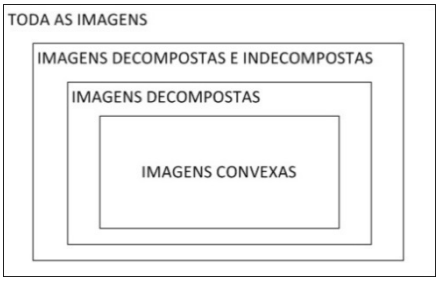

Figura 2.14 - Hierarquia dos conjuntos de imagens.
Fonte: (PARK; CHIN, 1995).

Existe uma infinidade de limitadores para as imagens côncavas, porém se as imagens forem restringidas ao tamanho 3x3, a quantidade é reduzida para vinte e oito tipos que são definidos por $\boldsymbol{Q_{Ti}}$. A variável $\boldsymbol{T}$ define o tipo do limitador, isto é,

$\boldsymbol{T = U, J, L, V, r}$, e $\boldsymbol{i}$ é a direção do início do código da cadeia, com $\boldsymbol{i = 0, 1, \ldots, 7}$ (Figura 2.15).

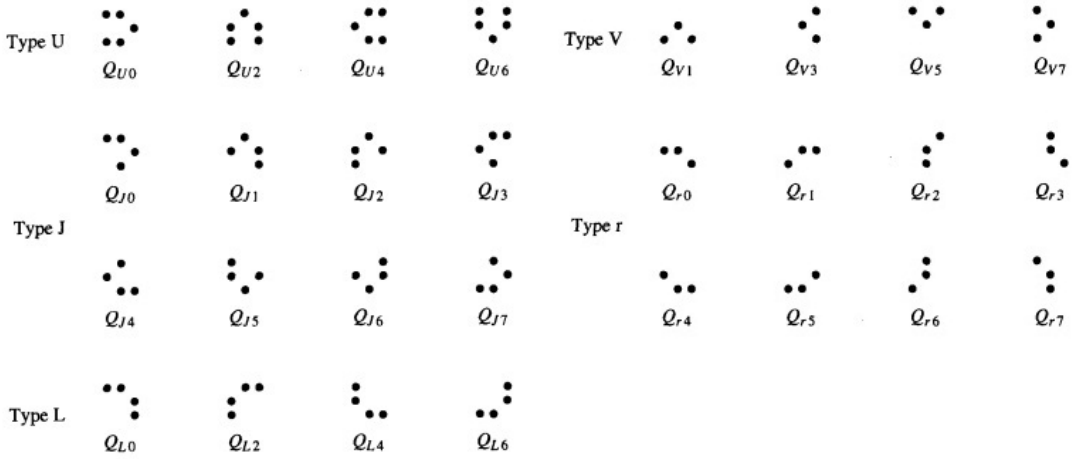

Figura 2.15 - Vinte e oito limitadores para imagens côncavas.
Fonte: (PARK; CHIN, 1995).

Analisando os vinte e oito limitadores através do código da cadeia, Park (PARK; CHIN, 1995) encontrou todos os fatores primos para imagens côncavas respeitando o tamanho 3x3 como é mostrado na Figura 2.16 e que estão representados utilizando-se o código da cadeia. Os expoentes indicam quantas vezes o valor é repetido.

Q_{U0}	Q_{L0}	$Q_{V1}Q_{V3}4^26^2$	Q_{r0}	Q_{r1}
$Q_{U0}0^22^24^2$	$Q_{L0}2^24^2$	$Q_{V1}Q_{V3}4^2Q_{V7}$	$Q_{r0}2^245$	$Q_{r1}24^2Q_{V7}$
$Q_{U0}0^2234$	$Q_{L0}234$	$Q_{V1}Q_{V3}456$	$Q_{r0}2^2Q_{V5}6$	$Q_{r1}24Q_{r6}$
$Q_{U0}0124^2$	Q_{V1}	$Q_{V1}Q_{V3}4Q_{r6}$	$Q_{r0}2^2Q_{r5}$	$Q_{r1}2Q_{V5}6^2$
$Q_{U0}0134$	$Q_{V1}2^24^2Q_{V7}$	$Q_{V1}Q_{V3}Q_{V5}6^2$	$Q_{r0}235$	$Q_{r1}2Q_{V5}Q_{V7}$
Q_{J0}	$Q_{V1}2^24Q_{r6}$	$Q_{V1}Q_{V3}Q_{V5}Q_{V7}$	$Q_{r0}2Q_{r4}6$	$Q_{r1}2Q_{r5}6$
$Q_{J0}02^24^2$	$Q_{V1}2^2Q_{V5}6^2$	$Q_{V1}Q_{V3}Q_{r5}6$	$Q_{r0}24^2$	$Q_{r1}25^2$
$Q_{J0}0234$	$Q_{V1}2^2Q_{V5}Q_{V7}$	$Q_{V1}Q_{V3}5^2$	$Q_{r0}Q_{V3}4^26$	$Q_{r1}346^2$
$Q_{J0}124^2$	$Q_{V1}2^2Q_{r5}6$	$Q_{V1}Q_{r3}46^2$	$Q_{r0}Q_{V3}45$	$Q_{r1}34Q_{V7}$
$Q_{J0}134$	$Q_{V1}2^25^2$	$Q_{V1}Q_{r3}4Q_{V7}$	$Q_{r0}Q_{V3}Q_{V5}6$	$Q_{r1}356$
Q_{J1}	$Q_{V1}234Q_{V7}$	$Q_{V1}Q_{r3}56$	$Q_{r0}Q_{V3}Q_{r5}$	$Q_{r1}3Q_{r6}$
$Q_{J1}2^24^26$	$Q_{V1}23Q_{r6}$	$Q_{V1}Q_{r3}Q_{r6}$	$Q_{r0}Q_{r3}5$	$Q_{r1}Q_{r4}6^2$
$Q_{J1}2^245$	$Q_{V1}2Q_{r4}6^2$	$Q_{V1}3^26^2$	$Q_{r0}Q_{r3}46$	$Q_{r1}Q_{r4}Q_{V7}$
$Q_{J1}2346$	$Q_{V1}2Q_{r4}Q_{V7}$	$Q_{V1}3^2Q_{V7}$	$Q_{r0}3^26$	$Q_{r1}4^26$
$Q_{J1}235$	$Q_{V1}24^26$	$Q_{V1}346$	$Q_{r0}34$	$Q_{r1}45$
	$Q_{V1}245$	$Q_{V1}35$		

Figura 2.16 - Todos os fatores primos côncavos.
Fonte: (PARK; CHIN, 1995).

$$Q_{r0}Q_{r1}2^3 35^2 Q_{r6}$$

Figura 2.17 - Exemplo de código da cadeia.
Fonte: (PARK; CHIN, 1995).

A Figura 2.17 mostra um exemplo na aplicação do código da cadeia. O sinal de + indica o início do código $Q_{r0}Q_{r1}2^3 35^2 Q_{r6}$ o qual faz uso dos limitadores para imagens côncavas conforme Figura 2.18, partindo do elemento Q_{r0} no sentido anti-horário até voltar no mesmo elemento.

Segundo Park (PARK; YOO, 2001), existem duas formas de se fazer a decomposição de um elemento S arbitrário (côncavo ou convexo), uma chamada de "Decomposição Forte" (mostrado anteriormente na equação 2.10) e outra denominada de "Decomposição Fraca" (como mostra a equação 2.12).

$$S = C_1 \cup C_2 \cup \ldots \cup C_M \quad \textbf{(2.9)}$$

A equação 2.13 mostra um elemento qualquer S decomposto pela união de um conjunto de elementos menores C_j de fator 3x3 e tal equação, assim como a equação 2.11, torna possível realizar a decomposição de uma imagem qualquer Xpor Sde modo que:

$$X \oplus S = (X \oplus C_1) \cup (X \oplus C_2) \cup \ldots \cup (X \oplus C_M) \quad \textbf{(2.10)}$$

Se cada elemento C_j da equação 2.13 for tratado independentemente como na equação 2.10, obtém-se outra equação (equação 2.14), chamada "Decomposição Híbrida", que é dado por:

$$X \oplus S = (X \oplus C_{11} \oplus C_{12} \oplus \ldots \oplus C_{1K}) \cup (X \oplus C_{21} \oplus C_{22} \oplus \ldots \oplus C_{2K}) \cup \ldots \cup (X \oplus C_{W1} \oplus C_{W2} \oplus \ldots \oplus C_{WK}) \quad \textbf{(2.11)}$$

Portanto, a "Decomposição Híbrida" é a forma mais geral de decomposição. Se $W = 1$, a equação 2.14 passa a ser igual a 2.10 (decomposição forte), e se $K = 1$, a equação 2.14 passa a ser igual a 2.13, tornando-se uma forma geral do processo de decomposição.

Observando todas as técnicas já citadas, Shih (SHIH; WU, 2005) desenvolveu um algoritmo em seu trabalho propondo uma técnica inovadora servindo de base para a elaboração deste trabalho. Na Figura 2.18 é mostrado o diagrama de blocos para o algoritmo de decomposição de Shih.

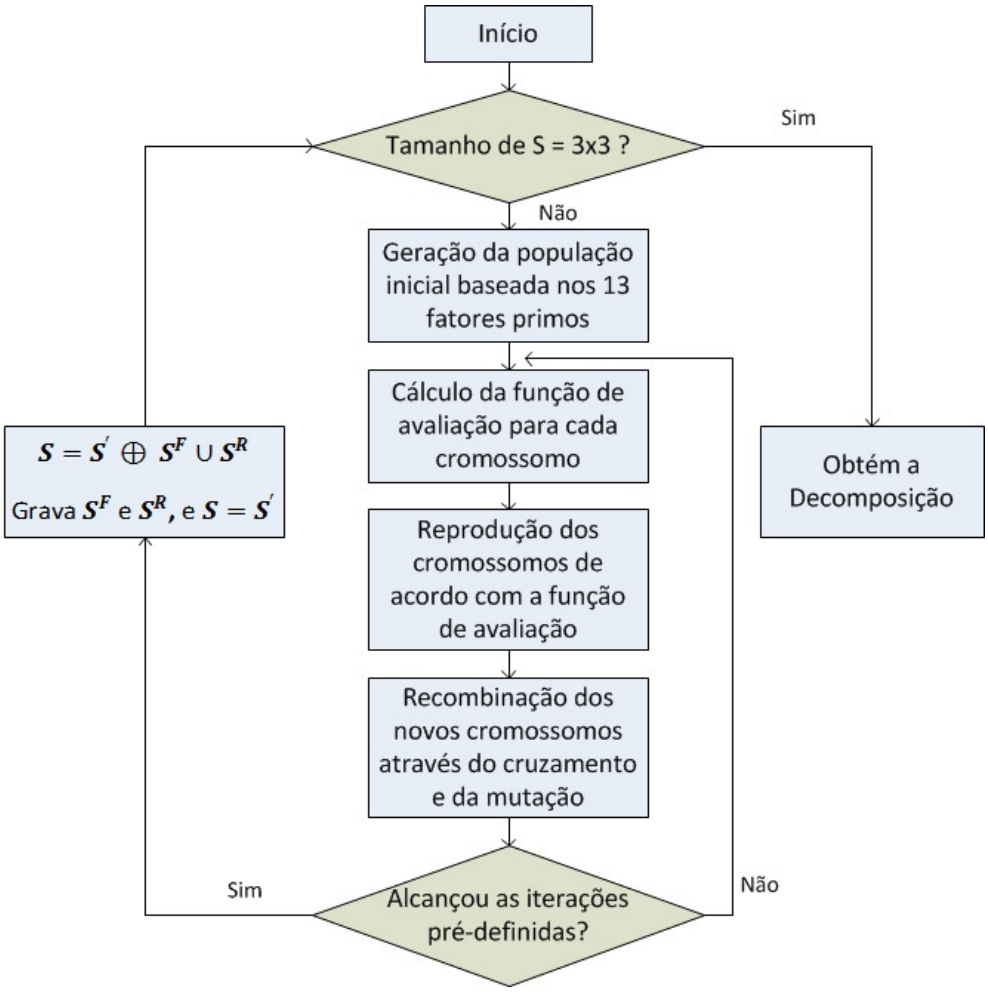

Figura 2.18 - Diagrama de blocos do algoritmo de decomposição

Fonte: Adaptado de (SHIH; WU, 2005).

O diagrama resume o algoritmo nas seguintes etapas:

1- Checagem do tamanho do elemento estruturante S, se for 3x3, vá para o passo 7.
2- População inicial baseada nos treze fatores primos é gerada.
3- Cálculo da função de avaliação para cada cromossomo.
4- Reprodução dos cromossomos usando cruzamento e mutação.
5- Se o número de iterações não foi atingido, volta para o passo 3.
6- Grava a decomposição encontrada na forma $\boldsymbol{S = S' \oplus S^F \cup S^R}$, onde $\boldsymbol{S'}$ e $\boldsymbol{S^F}$ são elementos estruturantes da decomposição e $\boldsymbol{S^R}$ é a diferença entre $\boldsymbol{S}$ e $\boldsymbol{S' \oplus S^F}$, e define $\boldsymbol{S = S'}$ e vá para o passo 1.
7- Saída da decomposição completa encontrada.

Como resultado, os experimentos de Shih (SHIH; WU, 2005) mostraram que a recursividade aliada aos algoritmos genéticos foram eficientes para a decomposição dos elementos estruturantes obtendo em menor tempo a decomposição em relação aos estudos demonstrados anteriormente.

2.5 Considerações Finais

Neste capítulo foi apresentada a revisão bibliográfica dos temas que envolvem este trabalho de mestrado. Foram abordados alguns tópicos da morfologia matemática, fornecendo alguns operadores que serão utilizados por este trabalho e que são muito utilizados para processamento de imagens. Anteriormente ao processo de decomposição, é necessária uma análise prévia dos métodos de decomposição do elemento estruturante que juntamente com os algoritmos genéticos diminuem o tempo de execução sempre buscando a otimização do resultado que é a decomposição.

Capítulo 3

METODOLOGIA PARA DECOMPOSIÇÃO DE ELEMENTOS ESTRUTURANTES EM MORFOLOGIA MATEMÁTICA

Neste capítulo, a metodologia desenvolvida é exposta mais detalhadamente. O capítulo está dividido da seguinte forma: A seção 3.1, descreve a metodologia do trabalho, onde são mostradas todas as regras adotadas para a elaboração do algoritmo. A seção 3.2, trata da implementação do algoritmo desenvolvido e são descritos os passos para a execução dessas regras.

3.1 Metodologia do Trabalho

Como citado anteriormente, o objetivo deste trabalho de mestrado é desenvolver uma otimização inteligente da decomposição de elemento estruturante usado em morfologia matemática binária e para fundamentar a proposta, foi utilizado o artigo de Shih (SHIH; WU, 2005) como base teórica.

O método utilizado para esta otimização, consiste primeiramente em dividir a imagem original (elemento estruturante) de tamanho 15x15 em 9 (nove) subimagens de tamanho 5x5, como pode ser visualizado nas Figuras 3.1 e 3.2.

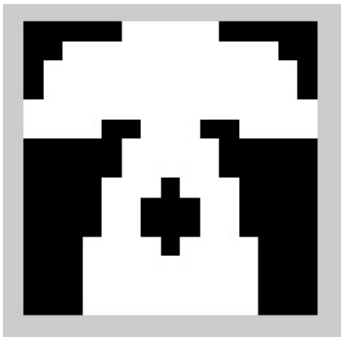

Figura 3.1 - Imagem Original 15x15.

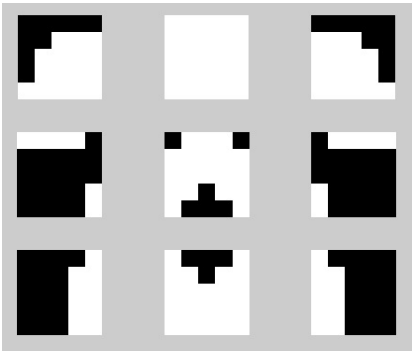

Figura 3.2 - Imagem Dividida em 9 subimagens 5x5.

Após a divisão, cada subimagem passa pelo algoritmo sequencialmente, uma após a outra, até que todas sejam processadas. Para cada subimagem é gerada uma quantidade de cromossomos definida previamente como população inicial. Cada cromossomo é composto por uma quantidade fixa de genes com elementos estruturantes codificados de tamanho 3x3 (Figura 3.3) e de uma sequência fixa de operações morfológicas de dilatação e união (duas e quatro respectivamente) seguindo o modelo proposto na Figura 3.4.

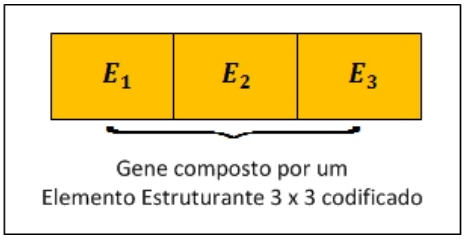

Figura 3.3 - Modelo de gene.

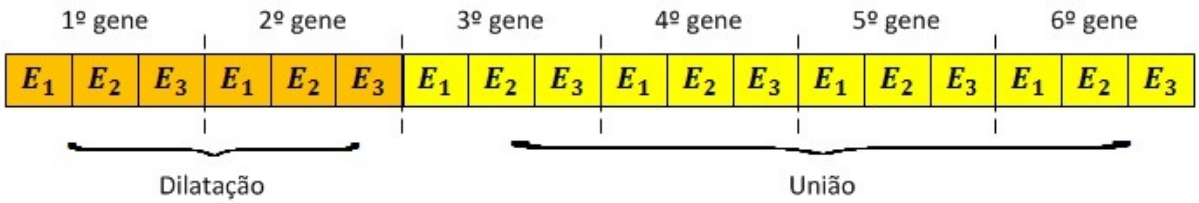

Figura 3.4 - Modelo do Cromossomo.

A codificação dos elementos estruturantes (matrizes de tamanho 3x3) em estruturas lineares, e a quantidade de operações morfológicas fixa, resulta na padronização do tamanho do cromossomo em seis genes, seguindo o padrão utilizado em todos os testes desse trabalho, onde a imagem original tem o tamanho 15x15. Essa padronização torna possível implementar o modelo proposto utilizando os conceitos abordados em *software* e principalmente em *hardware*.

A Figura 3.5 é composta de um elemento estruturante de tamanho 3x3 na forma de uma matriz, onde cada linha é composta por três números binários. A codificação consiste em transformar cada linha dessa matriz em um número decimal inteiro de 0 a 7 correspondente.

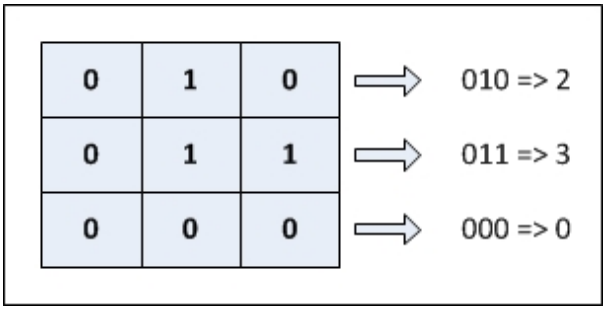

Figura 3.5 - Codificação do Elemento estruturante.

Após o elemento ser codificado, obtém-se uma estrutura linear composta de números inteiros formando cada gene do cromossomo (Figura 3.6). Isso reduz em um terço o espaço ocupado pelo elemento estruturante dentro do cromossomo passando de nove dígitos (binário) para três dígitos (decimal).

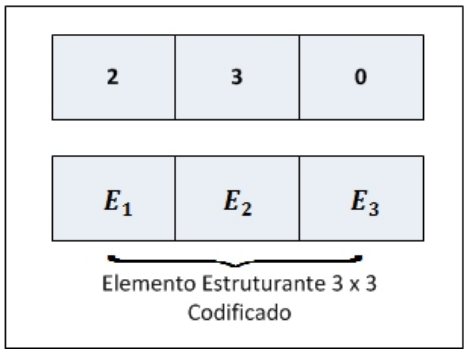

Figura 3.6 - Codificação do gene.

Para iniciar a implementação do algoritmo proposto, deve-se levar em consideração alguns pontos, sendo eles:

- A imagem original (Figura 3.1) será dividida em partes menores de tamanho 5x5 (Figura 3.2);

- O cromossomo (Figura 3.4) é composto de uma sequência fixa de seis estruturas iguais à Figura 3.6 para que, ao executar as operações morfológicas uma após a outra, chegue-se na imagem pretendida de tamanho 15x15;

- Para executar a operação de dilatação são necessários dois genes, onde sempre o primeiro gene é dilatado pelo segundo (Figura 3.7).

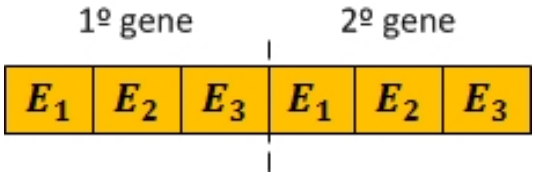

Figura 3.7 - Dilatação do 1º gene pelo 2º.

- Nos demais genes a operação de união é utilizada. Para que isso ocorra é necessário que a subimagem seja dividida em quadrantes (Figura 3.8), distribuídos pelos eixos x e y, com origem no centro.

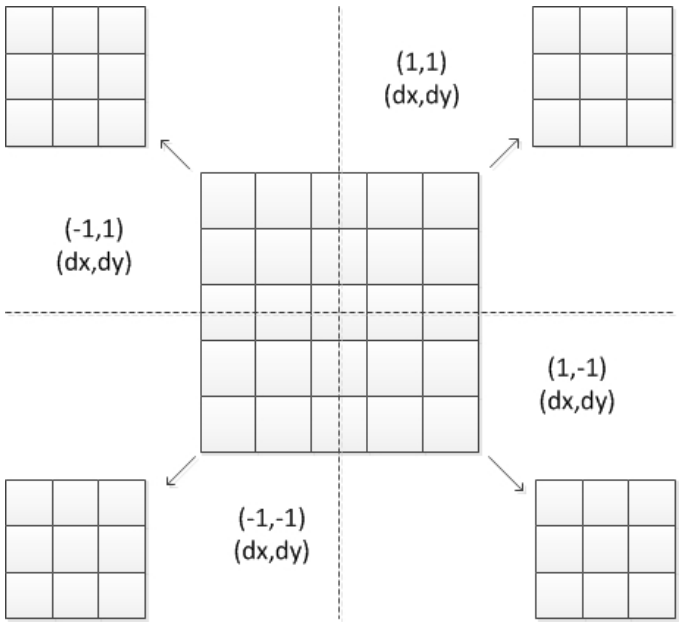

Figura 3.8 - Subimagem dividida em quadrantes.

Os deslocamentos para os eixos x (Dx) e y (Dy) são fixados em relação à origem podendo assim assumir um valor positivo ou negativo como ilustrado a seguir na Figura 3.9;

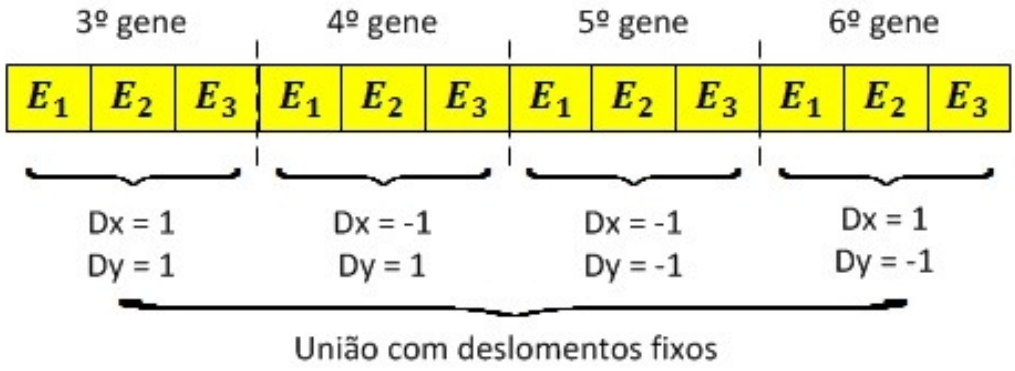

Figura 3.9 - Operações de união.

O comprimento do cromossomo fixado em seis genes é suficiente para que a decomposição encontrada gere novamente a imagem original. Isso se deve ao fato de que o elemento estruturante 3x3 ao ser dilatado uma única vez aumenta seu tamanho para 5x5 e também por que se dividirmos a subimagem em quatro quadrantes, as quatro uniões são suficientes para cobrir toda essa divisão se preenchendo assim toda a subimagem.

Ao término da geração dos cromossomos é feita a montagem das imagens correspondentes a cada cromossomo da população para que possa ser feito o cálculo da função de avaliação (menor erro) utilizado como critério de parada do algoritmo onde será obtida a decomposição desejada.

Com a avaliação realizada, são selecionados três indivíduos (cromossomos) ao acaso e dentre eles são escolhidos os dois mais aptos (com os menores erros em relação a imagem original) para que, utilizando o conceito de algoritmos genéticos, sejam executadas as operações de cruzamento e mutação e a substituição dos dois elementos menos aptos da população (com maiores erros). Após uma quantidade de iterações, denominada gerações, a função de avaliação ao encontrar um indivíduo com erro igual a zero encerra a busca e obtém a decomposição da subimagem.

3.2 Implementação

O algoritmo elaborado seguindo a ideias propostas será executado para que se encontre a decomposição de uma imagem qualquer de tamanho 15x15 conforme descrito abaixo e ilustrado pelo diagrama de blocos na Figura 3.10:

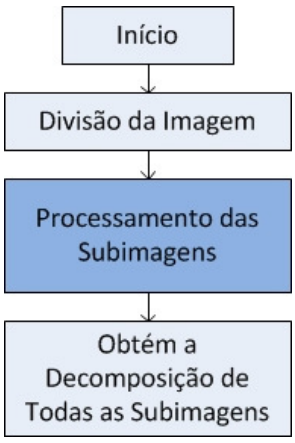

Figura 3.10 - Diagrama de Blocos Para Decomposição.

A. **Divisão da Imagem** – A imagem original 15x15 é subdividida em nove subimagens 5x5 para o processamento sequencial de todas as partes.

B. Processamento das Subimagens – Nesta parte do algoritmo, cada subimagem é processada separadamente uma após a outra descrita pelo pseudocódigo em alto nível (Figura 3.11) e um Fluxograma (Figura 3.12). Para que o conjunto de subimagens seja processado, são necessárias algumas atribuições de valores às variáveis do algoritmo. Na Tabela 3.1 são descritas as variáveis do algoritmo.

Tabela 3.1 - Variáveis de Inicialização.

Variáveis	Descrição
c	Nº de cromossomos
g	Nº de genes (*default* = 6)
a	Nº de elementos selecionados para a escolha dos 2 melhores utilizados no cruzamento/mutação (*default* = 3)
imp	Subimagem original para comparação de erro
Tc	Taxa de cruzamento (%)
Tm	Taxa de mutação (%)

```
procedimento AGEL
inicio
        %---- Definir valores iniciais para as variáveis ----%
        inicializar(c,g,a,imp,Tc,Tm)
        el <- [ ] %matriz que armazena a população inicial
        im <- [ ] %matriz que armazena as subimagens geradas
        e <- [ ] %vetor que armazena os erros calculados
        ct_g <- 1 %contador de gerações
        %---- População Inicial ----%
        função população(el,c,g) %função para criar a população inicial
        para i <- 1 até c
                inicio
                        função f_imagem(el) %função para montar as subimagens
                        função f_erro(imp,im) %função para cálculo do erro
                fim para
        %---- Evolução ----%
        enquanto e > 0
                função f_geracao(a,c,im,e,el,imp) %função para evolução da geração
                ct_g <- ct_g + 1 %contador de gerações incrementado de uma unidade
                fim enquanto
        fim procedimento
```

Figura 3.11 - Pseudocódigo em alto nível.

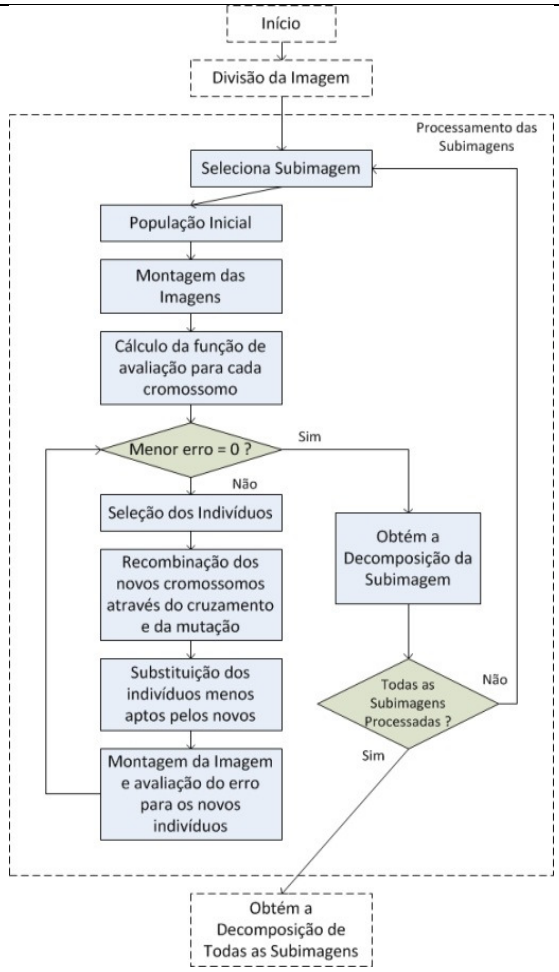

Figura 3.12 - Processamento das Subimagens.

1. **Seleciona Subimagem** – O processamento das subimagens inicia com a seleção de uma subimagem.

2. **Gerar população inicial** – A população inicial (número de cromossomos) é definida pelo usuário (Figura 3.4).

Por exemplo, supondo que a seguinte rede de cromossomos fosse gerada (Figura 3.13), cada linha dessa figura representaria um cromossomo.

6	4	2	1	2	0	6	5	3	4	3	2	7	7	0	1	5	3
2	3	2	1	2	0	1	5	3	4	2	2	2	4	7	7	0	3
7	4	5	3	7	6	6	2	3	7	0	2	1	3	5	2	4	1
2	7	1	0	2	4	5	2	1	4	6	2	4	2	0	6	0	2

Figura 3.13 - Exemplo de rede de cromossomos.

3. **Montar a imagem** – Para cada cromossomo da população criada, é feita a montagem da imagem seguindo as operações (dilatações e uniões) para que possa ser feita uma comparação com a imagem original.

O algoritmo interpreta cada linha separadamente e a converte em uma subimagem (Figura 3.14), executando cada operação (dilatação ou união) conforme a codificação do cromossomo.

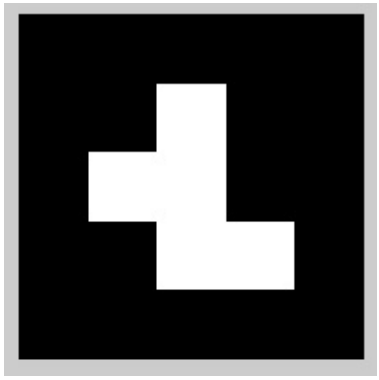

Figura 3.14 - Exemplo de Subimagem gerada para um cromossomo.

4. **Função de avaliação (Minimização do Erro)** – Na comparação de cada *pixel*, é calculado o erro da subimagem obtida em relação à subimagem original seguindo a equação 3.1:

$$\boldsymbol{Erro} = |\, \boldsymbol{Subimagem\ Original}\,(x,y) - \boldsymbol{Subimagem\ Obtida}(x,y)| \qquad \textbf{(3.1)}$$

Com a subimagem montada, é feita uma comparação com a subimagem original (Figura 3.15), a partir do cálculo do erro encontrado pela Equação 3.1.

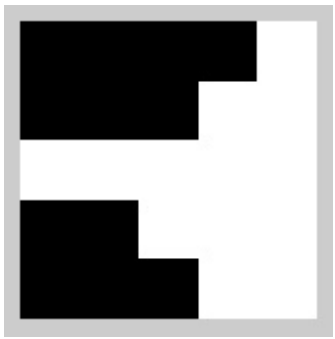

Figura 3.15 - Exemplo de Subimagem original.

5. **Teste #1** – Procura-se em todos os erros calculados para os cromossomos se existe um erro igual à zero. Caso exista, o ciclo termina e a execução avança para o passo 7. Caso não haja erro zero o algoritmo continua sua execução.

6. **Ciclo:**

6.1. Selecionar indivíduos – Após o teste do menor erro e a não solução do problema, são selecionados aleatoriamente três indivíduos da população e a partir desses são selecionados os dois melhores indivíduos (os dois com menor erro).

6.2. Cruzamento

6.2.1. Taxa de Cruzamento – A taxa de cruzamento (P_c) determina a probabilidade em que um cruzamento ocorrerá. Quanto maior for essa taxa, mais rapidamente novas estruturas serão introduzidas na população. Mas se essa for muito alta, a maior parte da população será substituída, e pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

Segundo Rezende (REZENDE, 2003), a taxa de cruzamento para os algoritmos genéticos deve ter as seguintes variações: $0,6 \leq P_c \leq 0,99$;

Segundo Obitko e Slavik (OBITKO; SLAVIK, 1999), a partir de estudos baseados em casos empíricos (na maioria dos casos com condição binária) a variação da taxa de cruzamento deve ser a seguinte: $0,8 \leq P_c \leq 0,95$;

Portanto, neste ponto do algoritmo é sorteado um número entre zero e um para saber se ocorrerá o cruzamento ou não, caso o número seja menor que o valor escolhido como taxa de cruzamento, dois novos filhos são gerados, caso contrário, a execução volta para o ponto 6.1. Neste passo também é sorteado qual o ponto de corte para se realizar o cruzamento.

6.3. Mutação

6.3.1. Taxa de Mutação – A taxa de mutação (P_m) determina a probabilidade na qual uma mutação deverá ocorrer. Uma baixa taxa de mutação previne que uma dada solução fique estagnada em um valor, causando uma convergência prematura, além de possibilitar que se chegue a qualquer ponto do espaço de busca. Com uma taxa muito alta, a busca se torna essencialmente aleatória. Os parâmetros genéticos afetam diretamente o desempenho dos AG's. Os efeitos decorrentes da escolha inadequada desses parâmetros vão desde o aumento no tempo de convergência, estagnação da busca, convergência prematura, maior necessidade de recursos computacionais até a não convergência para uma solução viável.

Segundo Rezende (REZENDE, 2003), a taxa de mutação para os algoritmos genéticos deve ter as seguintes varações: $0{,}001 \leq P_m \leq 0{,}1$;

Segundo Obitko e Slavik (OBITKO; SLAVIK, 1999), a partir de estudos baseados em casos empíricos (na maioria dos casos com condição binária) a variação da taxa de mutação deve ser: $0{,}005 \leq P_m \leq 0{,}01$;

Assim, baseado nos dados apresentados acima, para este trabalho as taxas de cruzamento e de mutação escolhidas foram as seguintes:

- Taxa de Mutação (P_m): 0,1 (10%);
- Taxa de Cruzamento (P_c): 0,9 (90%);

6.4. Substituição – Os dois novos indivíduos gerados pelo cruzamento e pela mutação são então inseridos na rede substituindo os dois indivíduos menos aptos da população.

6.5. Recálculo – É atualizada a imagem para os dois novos indivíduos igual o passo 3 e também a avaliação, passo 4, retornando para o passo 5.

7. **Obter Decomposição** – Com o erro igual zero, é selecionado o cromossomo obtendo assim a decomposição da subimagem.
8. **Teste #2** – Neste ponto, caso não tenha terminado o processamento de todas as subimagens, o algoritmo retorna à seleção de subimagem para que a próxima seja processada. Caso tenha terminado o processamento o algoritmo continua sua execução.

C. Decomposição Total – Após o processamento o algoritmo encontra a decomposição para todas as subimagens e termina sua execução.

Capítulo 4

RESULTADOS

Neste capítulo é descrito o estudo de caso, a partir do qual foi testado o algoritmo desenvolvido para decomposição de imagens, bem como todos os testes realizados. A divisão ocorre da seguinte forma: estudo de caso (seção 4.1) contendo seis subseções; 4.1.1 descreve os testes realizados; 4.1.2 apresentação das variáveis utilizadas pelo sistema; 4.1.3 destaca os critérios para execução do algoritmo; 4.1.4 apresenta os resultados dos testes realizados; 4.1.5 exemplo detalhado de execução do algoritmo e por fim, 4.1.6 comparação de resultados.

4.1 Estudo de Caso

Para efeito de comparação, foram usadas as mesmas seis imagens (Figura 4.1) do artigo base deste trabalho (SHIH; WU, 2005).

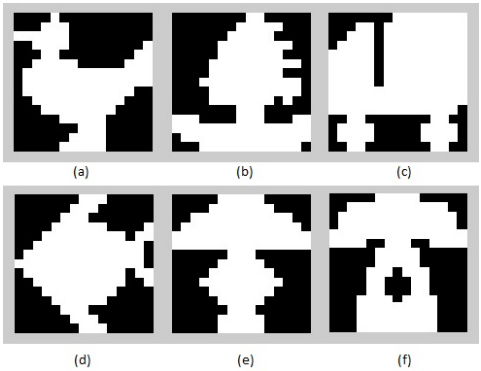

Figura 4.1 - Imagens utilizadas nos testes
(a) Pato (b) Barco (c) Caminhão (d) Peixe (e) Abajur (f) Telefone.

Para cada imagem o algoritmo é executado com o objetivo de definir o tempo necessário em segundos para que a imagem seja decomposta.

4.1.1 Testes Comparativos

Os testes foram realizados de duas maneiras:

- Sequencial;
- Paralelo.

No primeiro caso, o algoritmo desenvolvido foi executado de forma sequencial em três linguagens de programação diferentes (C, Matlab, Octave). No segundo, os testes consistiram em executar de forma paralela o algoritmo desenvolvido em duas linguagens de programação (C e Matlab).

Em todos os testes foram calculados o tempo médio e o desvio padrão para cada imagem e encontrados erros iguais a zero entre a imagem original e a imagem formada pelas decomposições encontradas. Para efeitos de comparação entre as linguagens, os programas em C e Matlab foram executados tanto após a compilação do programa quanto dentro do próprio Visual Studio 2012 e do Matlab 2013a respectivamente.

Nesse estudo de caso toda a bateria de testes foi executada utilizando um equipamento com as seguintes características de *hardware*:

- Processador: Intel Core I7 modelo 3770 3.4GHz;
- Memória RAM: DDR3 8GB 1333MHz;
- Placa Mãe: Gigabyte z77m-d3h
- Disco Rígido: Seagate 500GB 32MB Buffer Sata III

4.1.2 Variáveis Utilizadas

Para a realização dos testes deste estudo de caso, utilizaram-se variáveis pré-definidas (informadas pelo usuário), conforme Tabela 4.1. Os testes visaram alterar o valor do número de cromossomos da população (variável “c”) para comparação do tempo utilizado para encontrar o cromossomo mais apto para resolver a decomposição de cada imagem.

Tabela 4.1 - Variáveis utilizadas.

Variáveis	Valores
c	100 / 200 / 400
g	6
a	3
Tc	90%
Tm	10%

4.1.3 Critérios de Execução

Os testes com as seis imagens foram executados da seguinte forma: para cada uma das imagens, o algoritmo foi executado cinco vezes sem qualquer alteração, calculando o tempo médio e o desvio padrão ao final de cada execução e gravando os resultados. Esse processo foi repetido para cada troca de valor da variável “c” conforme Tabela 4.1. As escolhas desses valores como parâmetros para os testes foram obtidas após outros testes realizados com diferentes valores (maiores e menores) dos apresentados até o momento.

4.1.3.1 Sequencial

As Tabelas 4.2 e 4.3, assim como as Figuras 4.2 e 4.3, apresentam os resultados obtidos em segundos a partir da execução sequencial sobre cada uma das imagens (Figura 4.1). Os resultados obtidos estão divididos da seguinte forma:

- A Tabela 4.2 e a Figura 4.2 tratam da execução sem compilação do algoritmo, ou seja, sem a criação de um aplicativo executável, executando dentro da própria ferramenta utilizada para o desenvolvimento do código;
- A Tabela 4.3 e a Figura 4.3 tratam da execução com a compilação do algoritmo com exceção da Linguagem Octave, por apresentar “instabilidades” nos testes preliminares.

Tabela 4.2 - Tempo Médio (tm) e Desvio-Padrão (dp) para execução sequencial não compilado.

Estudo de Caso - Não Compilado		c = 100		c = 200		c = 400	
		tm (s)	dp (%)	tm (s)	dp (%)	tm (s)	dp (%)
C	ABAJUR	4,9420	4,7567	14,3520	10,1034	35,9160	2,7868
	BARCO	2,6060	0,4173	8,8280	2,4027	26,6084	3,9711
	CAMINHÃO	1,1340	0,1989	4,9480	1,5529	15,5460	1,8421
	PATO	2,9202	1,0004	12,9960	6,6949	35,7966	3,2618
	PEIXE	9,4860	6,4635	10,9630	2,1680	36,3122	2,1137
	TELEFONE	8,0286	12,8785	10,8372	2,9067	44,5332	25,1737
Matlab	ABAJUR	23,6391	25,8328	15,3570	3,2428	25,6969	2,0246
	BARCO	44,0764	57,4274	11,5294	1,3210	19,2406	0,5103
	CAMINHÃO	3,9876	0,3549	10,1024	8,0259	11,6238	1,0609
	PATO	11,7616	2,3759	16,2737	2,8588	26,6674	1,3042
	PEIXE	31,0413	37,1358	18,5099	2,9156	32,1243	3,2169
	TELEFONE	12,8028	6,3240	16,5861	3,8975	25,5464	2,5082
Octave	ABAJUR	37,9443	10,6986	55,1949	9,3726	81,0736	2,7083
	BARCO	28,8739	7,1830	42,6535	6,5959	68,0105	3,0284
	CAMINHÃO	25,2160	23,7590	23,8418	2,9279	38,4891	2,5000
	PATO	50,7451	19,5089	51,1014	4,6023	82,2651	6,3030
	PEIXE	44,8100	20,8759	63,3194	13,9723	89,0036	9,7180
	TELEFONE	28,1634	3,0462	54,0170	9,2132	80,7294	9,2667

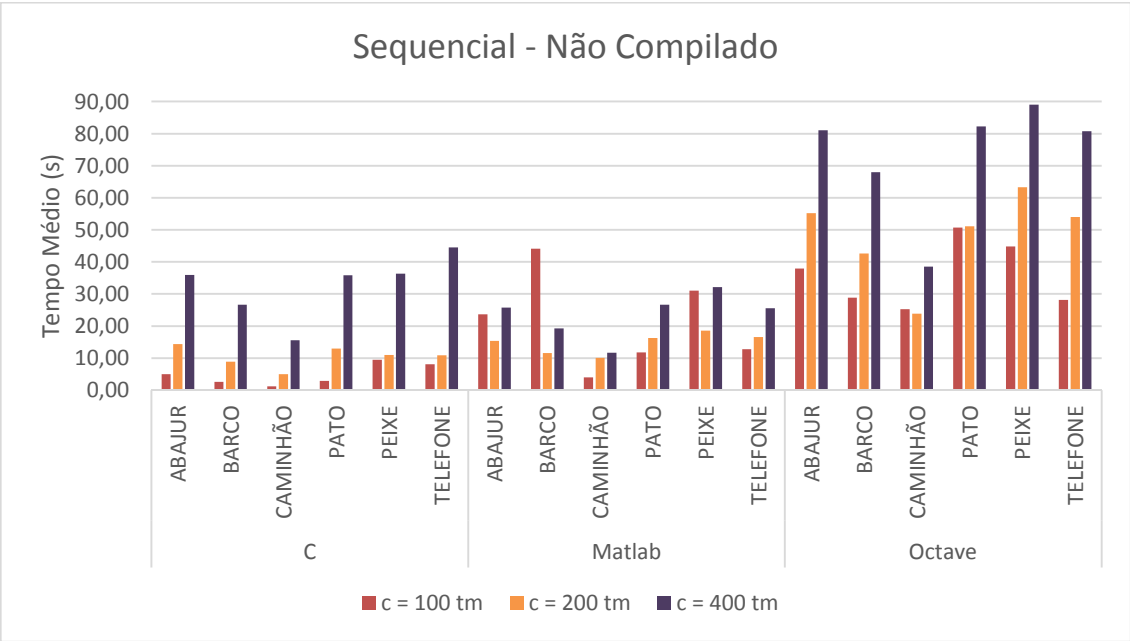

Figura 4.2 - Gráfico Sequencial não compilado.

Tabela 4.3 - Tempo Médio (tm) e Desvio-Padrão (dp) para execução sequencial compilado.

Estudo de Caso - Compilado		c = 100		c = 200		c = 400	
		tm (s)	dp (%)	tm (s)	dp (%)	tm (s)	dp (%)
C	ABAJUR	0,3880	0,0867	0,9300	0,1027	4,2180	0,3664
	BARCO	1,3420	1,3059	1,7000	1,5994	3,3940	0,4197
	CAMINHÃO	0,2360	0,0462	0,6280	0,0920	2,0200	0,2610
	PATO	0,3760	0,0945	1,4080	0,4832	4,1580	0,4036
	PEIXE	1,2140	1,2644	1,4720	0,1898	4,0320	0,3030
	TELEFONE	0,8620	0,6511	1,2720	0,1512	3,8000	0,3009
Matlab	ABAJUR	14,7189	7,7710	17,1018	2,7168	23,2971	1,5666
	BARCO	21,0204	30,3173	11,9657	0,8265	18,2728	2,1113
	CAMINHÃO	8,1056	6,5428	6,5512	0,4184	11,7029	0,7916
	PATO	17,2409	14,3457	17,0303	5,7561	23,5620	2,0968
	PEIXE	15,7281	7,2854	15,9009	3,9499	24,9589	2,2588
	TELEFONE	11,0020	3,0832	13,7695	3,1290	24,0483	1,7068

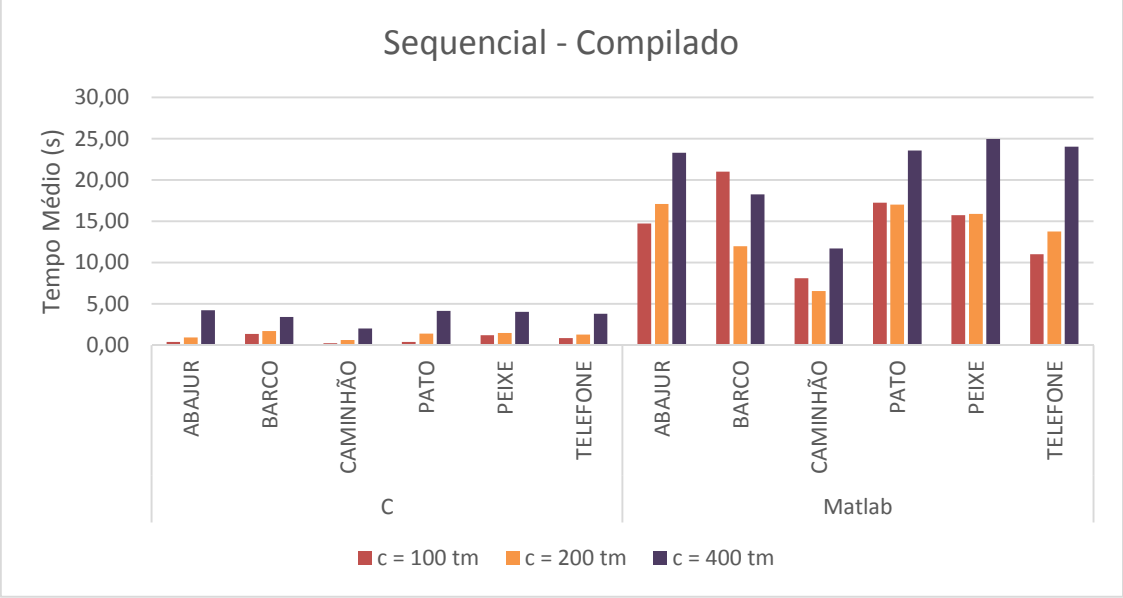

Figura 4.3 - Gráfico Sequencial compilado.

4.1.3.2 Paralelo

As Tabelas 4.4 e 4.5, assim como as Figuras 4.4 e 4.5, apresentam os resultados obtidos em segundos a partir da execução paralela sobre cada uma das imagens (Figura 4.1). Os resultados obtidos estão divididos da seguinte forma:

- A Tabela 4.4 e a Figura 4.4 tratam da execução sem compilação do algoritmo, ou seja, sem a criação de um aplicativo executável,

executando dentro da própria ferramenta utilizada para o desenvolvimento do código;

- A Tabela 4.5 e a Figura 4.5 tratam da execução com a compilação do algoritmo.

Tabela 4.4 - Tempo Médio (tm) e Desvio-Padrão (dp) para execução paralela não compilado.

Estudo de Caso – Não Compilado		c = 100		c = 200		c = 400	
		tm (s)	**dp (%)**	**tm (s)**	**dp (%)**	**tm (s)**	**dp (%)**
C	**ABAJUR**	2,2240	1,3917	3,9720	2,0664	8,3680	0,6889
	BARCO	2,2580	0,7981	3,1080	0,5775	6,8920	0,4474
	CAMINHÃO	0,6480	0,0926	1,9440	1,1847	3,8482	0,5013
	PATO	1,9260	0,9349	2,8440	0,1408	7,8640	0,3418
	PEIXE	4,4260	2,6673	5,0320	3,2357	12,8200	8,4370
	TELEFONE	2,2200	0,6252	3,6740	0,9137	8,5780	0,5883
Matlab	**ABAJUR**	7,8439	6,2324	5,6171	0,5042	8,6563	1,1901
	BARCO	2,7816	0,4791	3,8636	0,1928	7,3900	0,2559
	CAMINHÃO	5,3066	4,5587	3,2826	0,7077	5,6052	1,2293
	PATO	10,1806	6,4228	4,8888	1,2232	7,9437	1,5248
	PEIXE	5,4129	1,9332	5,2128	0,5009	8,7315	0,8152
	TELEFONE	5,4326	1,8916	5,6924	1,7326	7,9690	0,5913

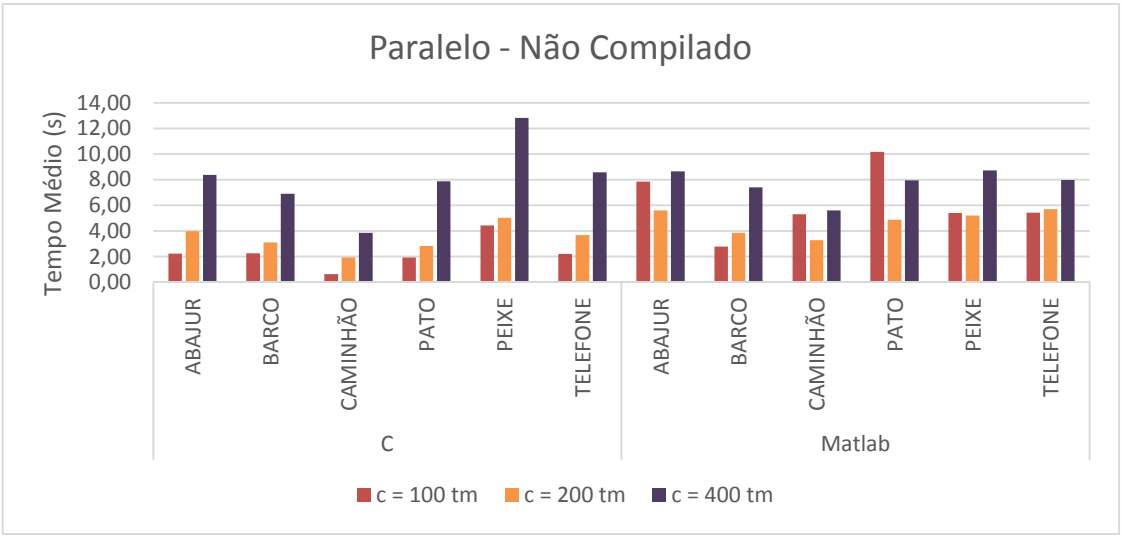

Figura 4.4 - Gráfico Paralelo não compilado.

Tabela 4.5 - Tempo Médio (tm) e Desvio-Padrão (dp) para execução paralela com compilação.

Estudo de Caso - Compilado		c = 100		c = 200		c = 400	
		tm (s)	dp (%)	tm (s)	dp (%)	tm (s)	dp (%)
C	ABAJUR	0,3822	0,2592	0,5260	0,3088	1,4240	0,4169
	BARCO	0,1860	0,1361	0,3880	0,0853	1,1440	0,0760
	CAMINHÃO	0,1020	0,0217	0,2340	0,0385	0,8460	0,0513
	PATO	0,9800	1,6414	0,3980	0,0785	1,1800	0,0543
	PEIXE	0,4240	0,1778	0,5500	0,4042	1,3220	0,1675
	TELEFONE	0,2800	0,1487	0,9100	1,1302	1,2000	0,0919
Matlab	ABAJUR	21,6607	36,2874	5,7345	0,7682	10,0175	0,8943
	BARCO	37,3000	41,8970	5,1370	0,3869	9,5731	0,6886
	CAMINHÃO	4,2657	2,3319	4,2588	0,1323	6,4983	0,8330
	PATO	24,0972	28,1266	7,0261	2,1399	10,2650	0,1144
	PEIXE	7,8647	7,8333	8,1542	3,5352	12,1366	2,7004
	TELEFONE	7,7395	6,1296	10,0921	5,7551	9,9236	1,6475

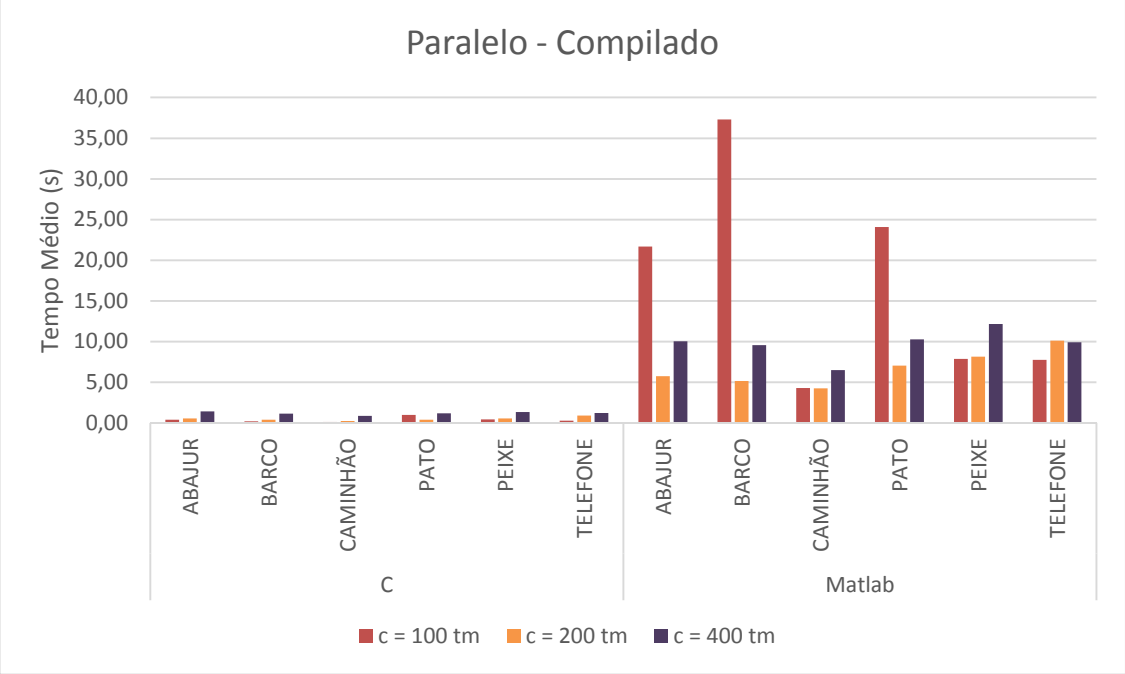

Figura 4.5 - Gráfico Paralelo compilado.

Com base nos dados contidos nas Tabelas 4.2 e 4.3 para o caso sequencial e nas Tabelas 4.4 e 4.5 para o caso paralelo, pode-se concluir que os melhores valores para a execução do algoritmo em ambas as formas foram encontrados utilizando o tamanho da população de cromossomos igual a 100 (variável 'c') e que a melhor linguagem utilizada nos testes foi a C executando após a compilação do algoritmo.

4.1.4 Discussão de Resultados

Para que a decomposição seja validada no modo sequencial, o algoritmo comparou cada subimagem de acordo com a função de custo escolhida passando para a próxima subimagem até que todas fossem processadas. Para o modo paralelo, a ordem de finalização do processamento varia de acordo com cada subimagem, pois o tempo varia de acordo com a complexidade das mesmas. Contudo, mesmo sendo finalizados fora da ordem que a imagem foi dividida, o algoritmo consegue reagrupar os pedaços para que a decomposição possa montar a imagem posteriormente.

4.1.4.1 Abajur

A primeira imagem utilizada nos testes realizados contém um objeto no formato de um abajur conforme Figura 4.6.

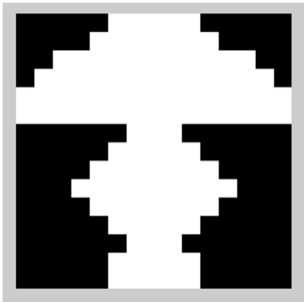

Figura 4.6 - Abajur.

Conforme descrito anteriormente, a imagem escolhida para o teste deve ser subdividida para o processamento da mesma (Figura 4.7).

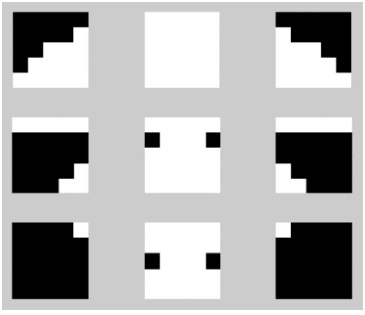

Figura 4.7 - Abajur subdividido.

A subdivisão facilita o processo de execução do algoritmo tanto sequencialmente como paralelamente pois no primeiro caso cada subimagem (1/9 da imagem original com resolução 15x15, ou seja, 5x5) será processado um após o outro e no segundo caso, ao mesmo tempo.

Figura 4.8 - "Pedaços" Originais do abajur.

Para todas as execuções do algoritmo, os "pedaços" originais (Figura 4.8) foram encontrados com erro igual a zero (Figura 4.9), porém, em cada execução e/ou linguagem pode haver um resultado diferente que leve ao mesmo objetivo (decomposição).

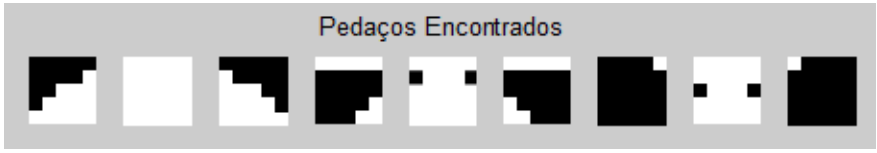

Figura 4.9 - "Pedaços" Encontrados do abajur.

Como ilustração, a Tabela 4.6 contém os cromossomos obtidos para os 9 pedaços com execução sequencial do algoritmo em Matlab e população igual a 100.

Tabela 4.6 - Rede de Cromossomos para o Abajur.

pedaço 1	0 1 3	1 1 6	0 0 5	0 0 1	0 3 5	6 4 7
pedaço 2	6 3 6	7 5 4	5 2 2	5 7 6	5 2 0	3 1 7
pedaço 3	3 0 7	0 0 4	0 0 4	0 4 3	7 7 5	4 6 7
pedaço 4	5 0 0	7 0 0	7 0 0	6 0 0	0 0 0	0 1 3
pedaço 5	0 7 6	2 4 7	7 4 3	7 2 5	6 6 0	5 2 7
pedaço 6	5 0 0	3 0 0	2 0 0	7 0 0	0 4 6	0 0 0
pedaço 7	1 0 0	1 0 0	1 0 0	0 0 0	0 0 0	0 0 0
pedaço 8	4 2 7	2 6 2	7 7 4	5 6 3	2 5 7	2 5 3
pedaço 9	0 0 0	3 0 4	0 0 0	4 0 0	0 0 0	0 0 0
	gene 1	**gene 2**	**gene 3**	**gene 4**	**gene 5**	**gene 6**
	dilatação		**união**	**união**	**união**	**união**

A Tabela 4.7 mostra a média dos tempos (em segundos) de todas as execuções nas linguagens escolhidas para desenvolvimento do algoritmo de modo sequencial de acordo com as quantidades de cromossomos escolhidas (n_crom). Em seguida, a Figura 4.10 mostra o gráfico da Tabela 4.7.

Tabela 4.7 - Abajur (Sequencial).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab	Octave
100	0,3880	4,9420	14,7189	23,6391	37,9443
200	0,9300	14,3520	17,1018	15,3570	55,1949
400	4,2180	35,9160	23,2971	25,6969	81,0736

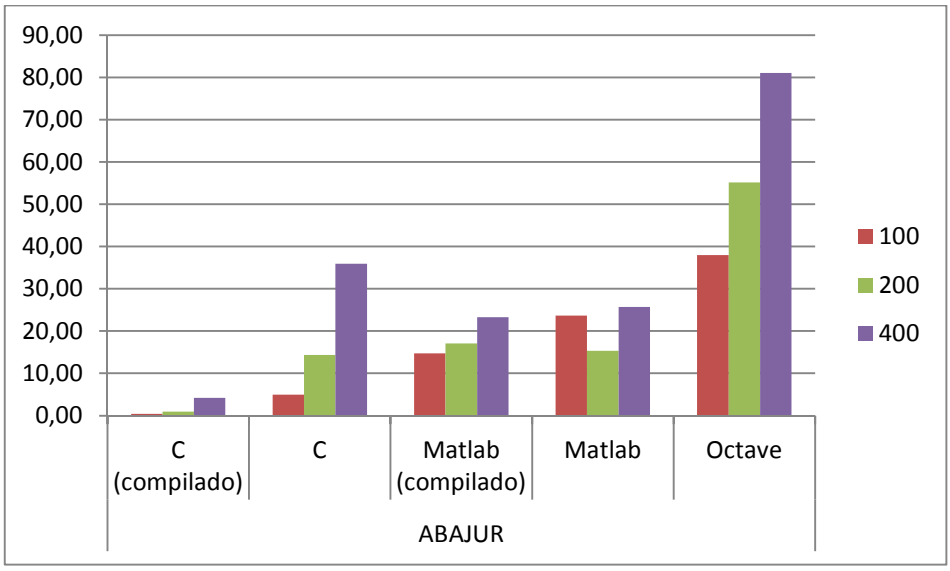

Figura 4.10 - Abajur (Sequencial).

Da mesma forma, a Tabela 4.8 e a Figura 4.11 retratam os resultados para o modo paralelo.

Tabela 4.8 - Abajur (Paralelo).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab
100	0,3822	2,2240	8,5103	7,8439
200	0,5260	3,9720	5,7345	5,6171
400	1,4240	8,3680	10,0175	8,6563

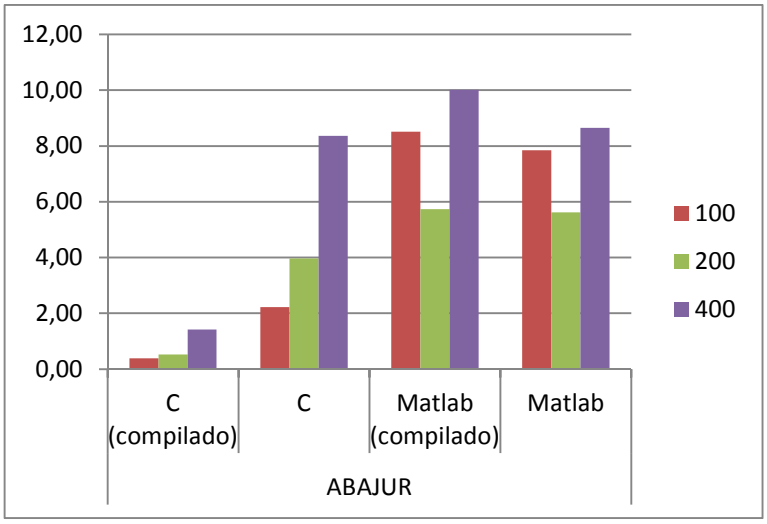

Figura 4.11 - Abajur (Paralelo).

4.1.4.2 Barco

A segunda imagem utilizada nos testes realizados contém um objeto no formato de um barco conforme Figura 4.12.

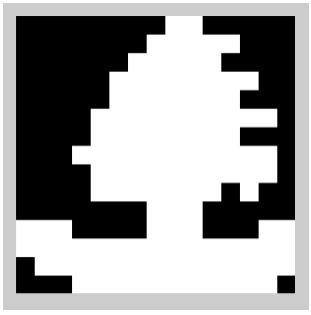

Figura 4.12 - Barco.

A imagem escolhida deve ser subdividida para o processamento (Figura 4.13).

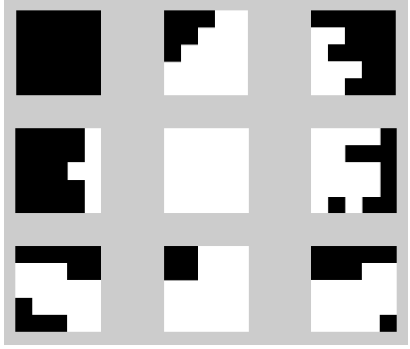

Figura 4.13 - Barco subdividido.

Para todas as execuções do algoritmo, os “pedaços” originais (Figura 4.14) foram encontrados com erro igual a zero (Figura 4.15).

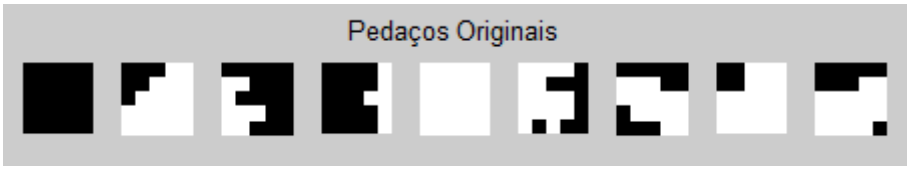

Figura 4.14 - “Pedaços” Originais do barco.

Figura 4.15 - “Pedaços” Encontrados do barco.

Como ilustração, a Tabela 4.9 contém os cromossomos obtidos para os 9 pedaços com execução sequencial do algoritmo em Matlab e população igual a 100.

Tabela 4.9 - Rede de Cromossomos para o Barco.

pedaço 1	0 0 0	0 2 0	0 0 0	0 0 0	0 0 0	0 0 0
pedaço 2	1 0 7	2 2 7	1 7 4	0 1 0	3 4 7	6 1 0
pedaço 3	4 4 0	0 4 4	0 0 0	0 6 4	0 7 6	0 0 0
pedaço 4	1 1 1	0 1 0	1 1 3	0 0 0	0 0 0	2 0 1
pedaço 5	7 6 5	5 1 6	0 2 6	2 5 0	1 0 5	5 3 1
pedaço 6	5 4 4	6 0 4	6 0 2	5 4 1	7 7 1	6 6 0
pedaço 7	0 7 2	4 6 1	0 0 6	0 1 5	1 2 0	3 2 3
pedaço 8	1 3 5	3 4 7	7 3 3	0 0 3	5 1 3	0 5 3
pedaço 9	0 3 5	0 5 2	0 3 7	0 0 0	7 7 5	7 6 4
	gene 1	**gene 2**	**gene 3**	**gene 4**	**gene 5**	**gene 6**
	dilatação		**união**	**união**	**união**	**união**

A Tabela 4.10 mostra a média dos tempos (em segundos) de todas as execuções nas linguagens escolhidas para desenvolvimento do algoritmo de modo sequencial de acordo com as quantidades de cromossomos escolhidas (n_crom). Em seguida, a Figura 4.16 mostra o gráfico da Tabela 4.10.

Tabela 4.10 - Barco (Sequencial).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab	Octave
100	1,3420	2,6060	21,0204	44,0764	28,8739
200	1,7000	8,8280	11,9657	11,5294	42,6535
400	3,3940	26,6084	18,2728	19,2406	68,0105

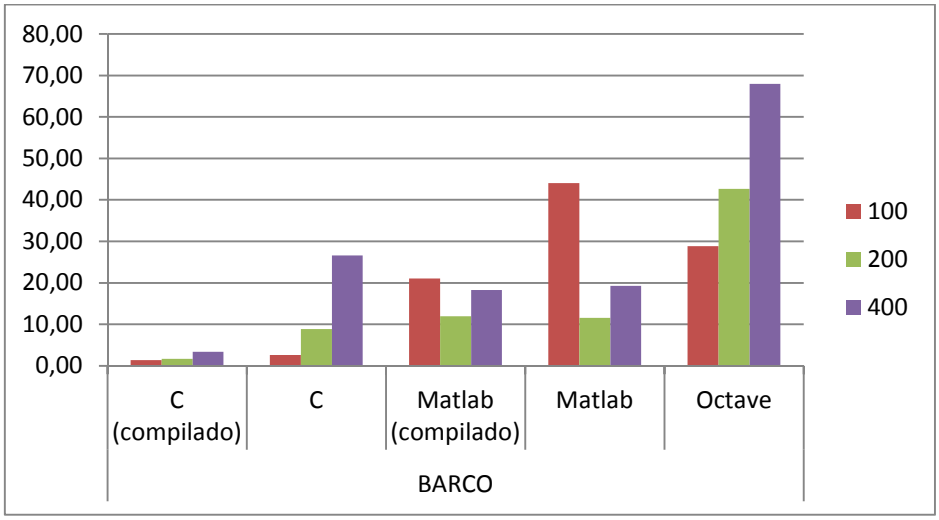

Figura 4.16 - Barco (Sequencial).

Da mesma forma, a Tabela 4.11 e a Figura 4.17 retratam os resultados para o modo paralelo.

Tabela 4.11 - Barco (Paralelo).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab
100	0,1860	2,2580	8,3360	2,7816
200	0,3880	3,1080	5,1370	3,8636
400	1,1440	6,8920	9,5731	7,3900

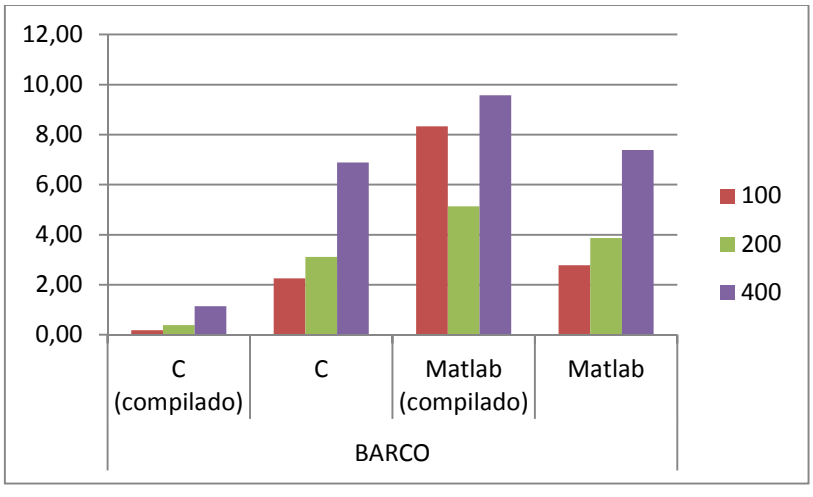

Figura 4.17 - Barco (Paralelo).

4.1.4.3 Caminhão

A terceira imagem utilizada nos testes realizados contém um objeto no formato de um caminhão conforme Figura 4.18.

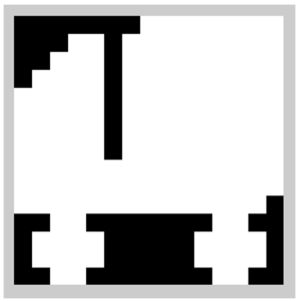

Figura 4.18 - Caminhão.

A imagem escolhida deve ser subdividida para o processamento (Figura 4.19).

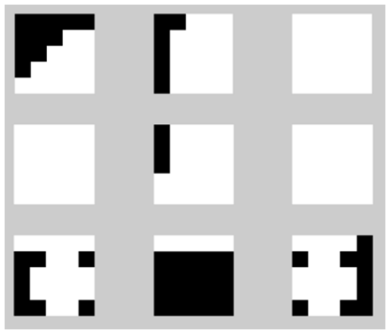

Figura 4.19 - Caminhão subdividido.

Para todas as execuções do algoritmo, os “pedaços” originais (Figura 4.20) foram encontrados com erro igual a zero (Figura 4.21).

Figura 4.20 - “Pedaços” Originais do caminhão.

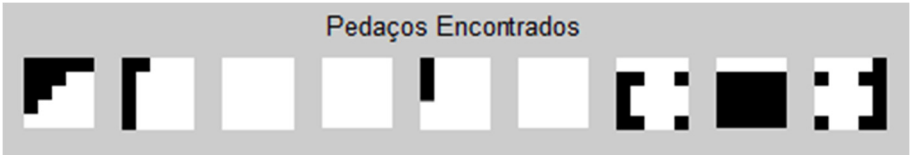

Figura 4.21 - "Pedaços" Encontrados do Caminhão.

Como ilustração, a Tabela 4.12 contém os cromossomos obtidos para os 9 pedaços com execução sequencial do algoritmo em Matlab e população igual a 100.

Tabela 4.12 - Rede de Cromossomos para o Caminhão.

pedaço 1	0 1 3	3 3 6	0 2 0	0 0 0	0 2 4	3 0 7
pedaço 2	3 3 2	1 5 7	2 2 2	1 2 0	3 2 2	5 5 3
pedaço 3	1 5 5	7 7 6	6 0 3	7 0 3	4 0 2	3 0 5
pedaço 4	4 7 7	7 5 3	7 0 4	7 4 0	3 1 7	4 0 0
pedaço 5	1 4 4	3 3 7	0 0 4	3 3 2	0 4 2	3 3 7
pedaço 6	3 2 3	7 1 7	1 2 1	4 4 0	7 6 5	7 7 7
pedaço 7	3 0 1	1 2 2	3 2 5	7 0 0	2 3 0	7 3 6
pedaço 8	7 0 0	5 0 0	0 0 0	7 0 0	0 0 0	0 0 0
pedaço 9	4 0 4	5 3 3	6 4 4	2 2 1	1 4 2	2 6 4
	gene 1	**gene 2**	**gene 3**	**gene 4**	**gene 5**	**gene 6**
	dilatação		**união**	**união**	**união**	**união**

A Tabela 4.13 mostra a média dos tempos (em segundos) de todas as execuções nas linguagens escolhidas para desenvolvimento do algoritmo de modo sequencial de acordo com as quantidades de cromossomos escolhidas (n_crom). Em seguida, a Figura 4.22 mostra o gráfico da Tabela 4.13.

Tabela 4.13 - Caminhão (Sequencial).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab	Octave
100	0,2360	1,1340	8,1056	3,9876	25,2160
200	0,6280	4,9480	6,5512	10,1024	23,8418
400	2,0200	15,5460	11,7029	11,6238	38,4891

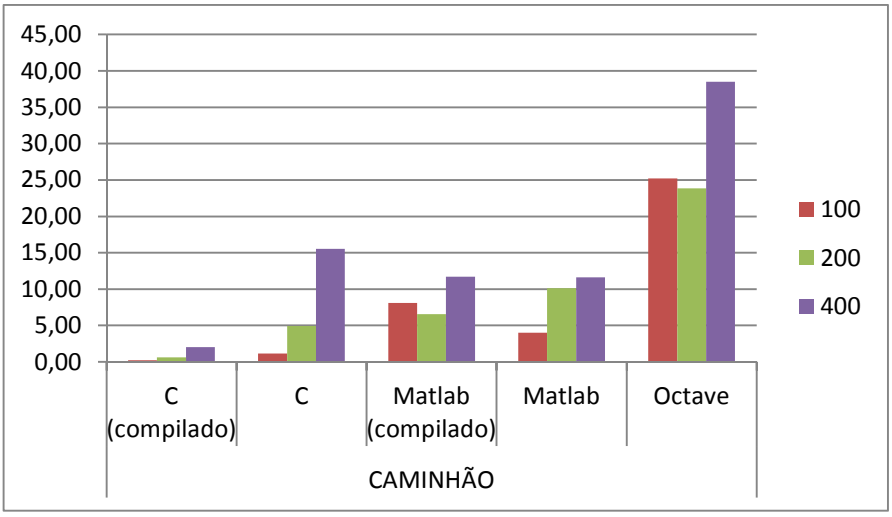

Figura 4.22 - Caminhão (Sequencial).

Da mesma forma, a Tabela 4.14 e a Figura 4.23 retratam os resultados para o modo paralelo.

Tabela 4.14 - Caminhão (Paralelo).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab
100	0,1020	0,6480	4,2657	5,3066
200	0,2340	1,9440	4,2588	3,2826
400	0,8460	3,8482	6,4983	5,6052

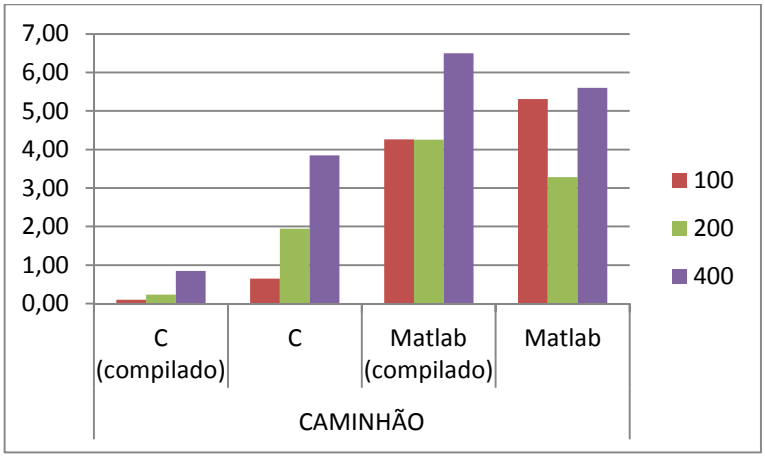

Figura 4.23 - Caminhão (Paralelo).

4.1.4.4 Pato

A quarta imagem utilizada nos testes realizados contém um objeto no formato de um abajur conforme Figura 4.24.

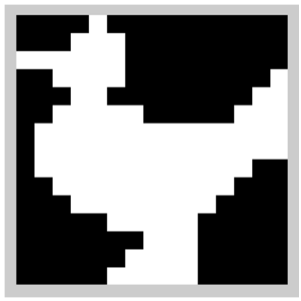

Figura 4.24 - Pato.

Conforme descrito anteriormente, a imagem escolhida para o teste deve ser subdividida para o processamento da mesma (Figura 4.25).

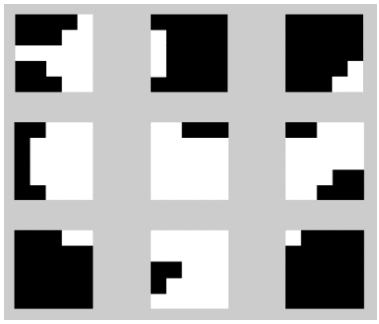

Figura 4.25 - Pato subdividido.

Para todas as execuções do algoritmo, os "pedaços" originais (Figura 4.26) foram encontrados com erro igual a zero (Figura 4.27).

Figura 4.26 - "Pedaços" Originais do pato.

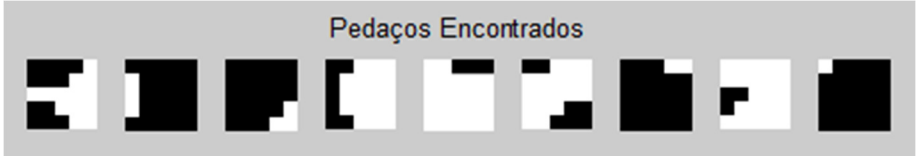

Figura 4.27 - "Pedaços" Encontrados do pato.

Como ilustração, a Tabela 4.15 contém os cromossomos obtidos para os 9 pedaços com execução sequencial do algoritmo em Matlab e população igual a 100.

Tabela 4.15 - Rede de Cromossomos para o Pato.

pedaço 1	1 3 0	1 2 3	0 1 2	0 0 6	7 1 0	4 0 3
pedaço 2	0 0 0	3 0 4	0 0 0	0 4 0	4 4 0	0 0 0
pedaço 3	0 0 2	0 0 1	0 0 0	0 0 0	0 0 0	0 1 3
pedaço 4	7 3 0	0 1 1	7 1 6	0 3 3	0 3 1	1 5 7
pedaço 5	6 0 5	4 7 7	0 5 5	2 4 2	0 2 3	6 7 5
pedaço 6	3 6 6	3 4 0	5 7 5	0 6 6	7 7 6	5 4 0
pedaço 7	0 0 0	0 3 5	3 0 0	0 0 0	0 0 0	0 0 0
pedaço 8	3 5 1	3 1 3	0 4 5	7 5 1	0 2 6	6 4 5
pedaço 9	0 0 0	6 1 0	0 0 0	4 0 0	0 0 0	0 0 0
	gene 1	**gene 2**	**gene 3**	**gene 4**	**gene 5**	**gene 6**
	dilatação		**união**	**união**	**união**	**união**

A Tabela 4.16 mostra a média dos tempos (em segundos) de todas as execuções nas linguagens escolhidas para desenvolvimento do algoritmo de modo sequencial de acordo com as quantidades de cromossomos escolhidas (n_crom). Em seguida, a Figura 4.28 mostra o gráfico da Tabela 4.16.

Tabela 4.16 - Pato (Sequencial).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab	Octave
100	0,3760	2,9202	17,2409	11,7616	50,7451
200	1,4080	12,9960	17,0303	16,2737	51,1014
400	4,1580	35,7966	23,5620	26,6674	82,2651

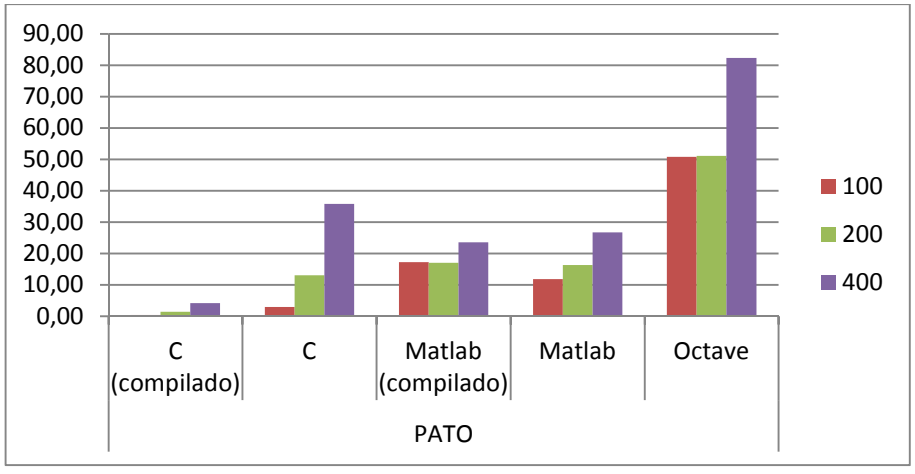

Figura 4.28 - Pato (Sequencial).

Da mesma forma, a Tabela 4.17 e a Figura 4.29 retratam os resultados para o modo paralelo.

Tabela 4.17 - Pato (Paralelo).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab
100	0,9800	1,9260	24,0972	10,1806
200	0,3980	2,8440	7,0261	4,8888
400	1,1800	7,8640	10,2650	7,9437

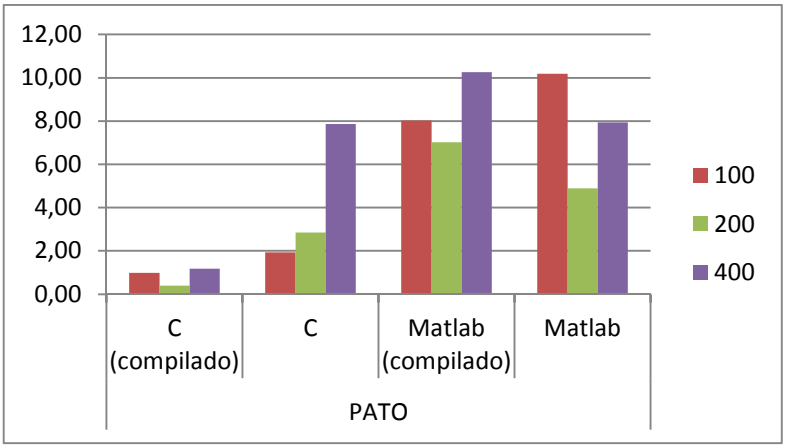

Figura 4.29 - Pato (Paralelo).

4.1.4.5 Peixe

A quinta imagem utilizada nos testes realizados contém um objeto no formato de um abajur conforme Figura 4.30.

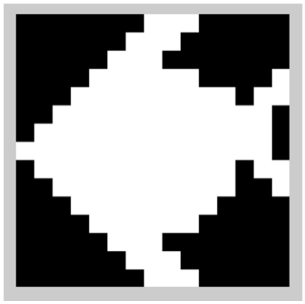

Figura 4.30 - Peixe.

Conforme descrito anteriormente, a imagem escolhida para o teste deve ser subdividida para o processamento da mesma (Figura 4.31).

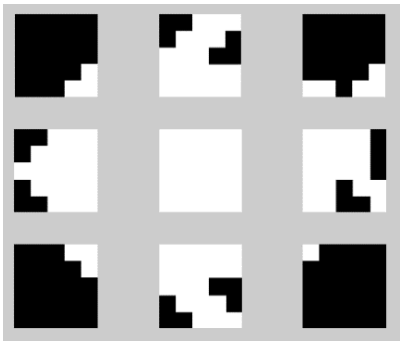

Figura 4.31 - Peixe subdividido.

Para todas as execuções do algoritmo, os “pedaços” originais (Figura 4.32) foram encontrados com erro igual a zero (Figura 4.33).

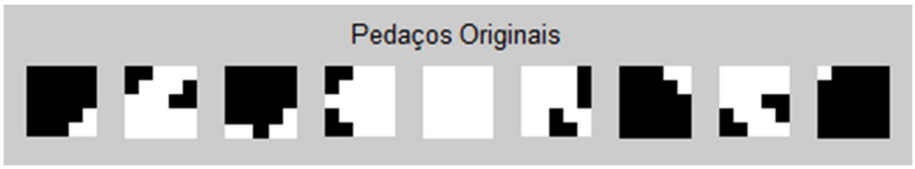

Figura 4.32 - “Pedaços” Originais do peixe.

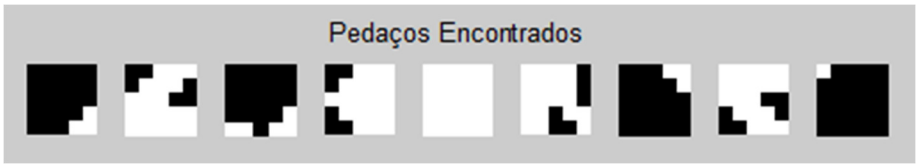

Figura 4.33 - “Pedaços” Encontrados do peixe.

Como ilustração, a Tabela 4.18 contém os cromossomos obtidos para os 9 pedaços com execução sequencial do algoritmo em Matlab e população igual a 100.

Tabela 4.18 - Rede de Cromossomos para o Peixe.

pedaço 1	0 0 1	0 0 1	0 0 0	0 0 0	0 0 0	0 1 3
pedaço 2	0 7 6	2 4 5	7 6 4	1 3 5	6 5 6	0 1 5
pedaço 3	0 0 1	0 1 2	0 0 0	0 0 0	0 0 6	0 1 1
pedaço 4	1 3 3	7 5 1	4 7 7	1 3 7	5 2 1	2 3 3
pedaço 5	7 4 3	6 1 7	7 1 4	4 3 1	2 3 7	1 2 7
pedaço 6	6 4 2	3 1 4	6 2 0	4 5 5	2 2 4	0 1 1
pedaço 7	0 0 0	4 0 1	3 1 0	0 0 0	0 0 0	0 0 0
pedaço 8	4 4 2	2 4 2	7 3 0	7 7 2	6 3 1	0 2 7
pedaço 9	0 0 0	0 1 0	0 0 0	4 0 0	0 0 0	0 0 0
	gene 1	**gene 2**	**gene 3**	**gene 4**	**gene 5**	**gene 6**
	dilatação		**união**	**união**	**união**	**união**

A Tabela 4.19 mostra a média dos tempos (em segundos) de todas as execuções nas linguagens escolhidas para desenvolvimento do algoritmo de modo sequencial de acordo com as quantidades de cromossomos escolhidas (n_crom). Em seguida, a Figura 4.34 mostra o gráfico da Tabela 4.19.

Tabela 4.19 - Peixe (Sequencial).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab	Octave
100	1,2140	9,4860	15,7281	31,0413	44,8100
200	1,4720	10,9630	15,9009	18,5099	63,3194
400	4,0320	36,3122	24,9589	32,1243	89,0036

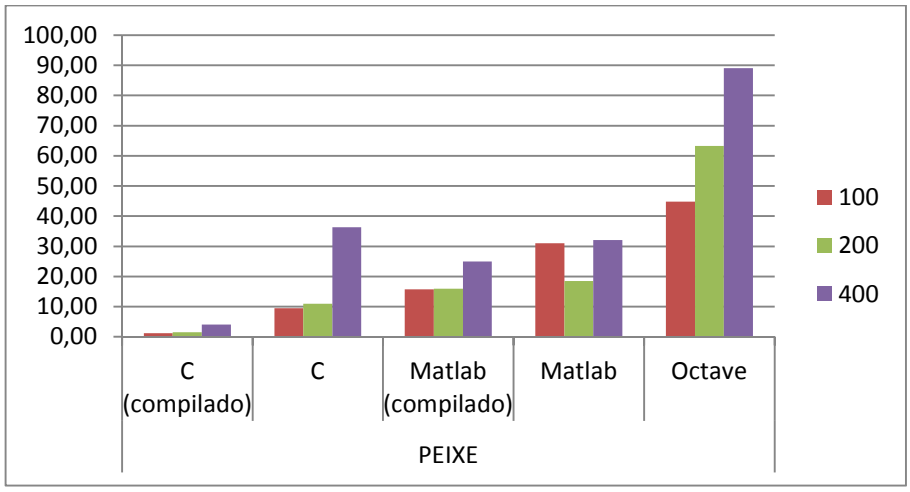

Figura 4.34 - Peixe (Sequencial).

Da mesma forma, a Tabela 4.20 e a Figura 4.35 retratam os resultados para o modo paralelo.

Tabela 4.20 - Peixe (Paralelo).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab
100	0,4240	4,4260	7,8647	5,4129
200	0,5500	5,0320	8,1542	5,2128
400	1,3220	12,8200	12,1366	8,7315

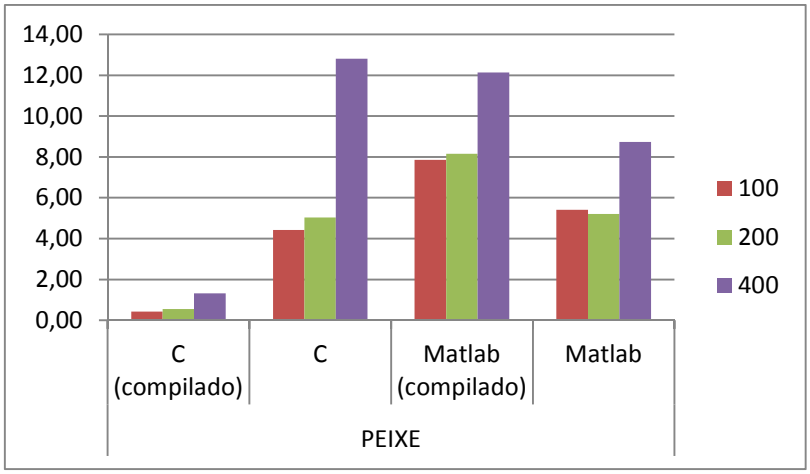

Figura 4.35 - Peixe (Paralelo).

4.1.4.6 Telefone

A sexta e última imagem utilizada nos testes realizados contém um objeto no formato de um abajur conforme Figura 4.36.

Figura 4.36 - Telefone.

Conforme descrito anteriormente, a imagem escolhida para o teste deve ser subdividida para o processamento da mesma (Figura 4.37).

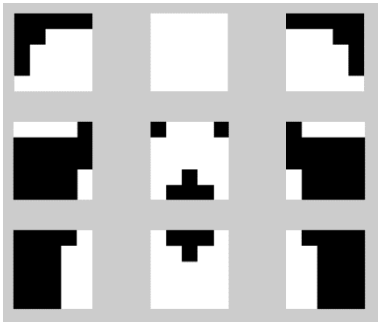

Figura 4.37 - Telefone subdividido.

Para todas as execuções do algoritmo, os “pedaços” originais (Figura 4.38) foram encontrados com erro igual a zero (Figura 4.39).

Figura 4.38 - “Pedaços” Originais do abajur.

Figura 4.39 - “Pedaços” Encontrados do abajur.

Como ilustração, a Tabela 4.21 contém os cromossomos obtidos para os 9 pedaços com execução sequencial do algoritmo em Matlab e população igual a 100.

Tabela 4.21 - Rede de Cromossomos para o Telefone.

pedaço 1	1 3 6	0 3 3	0 4 1	0 0 0	3 2 7	5 3 1
pedaço 2	7 5 3	5 7 7	5 2 3	6 5 3	5 5 6	5 0 6
pedaço 3	4 7 4	0 6 6	0 4 4	0 3 2	6 1 3	0 6 7
pedaço 4	0 0 0	7 1 1	2 0 0	7 0 0	0 0 0	0 1 1
pedaço 5	7 3 4	0 6 0	6 7 4	2 7 1	6 6 4	5 3 1
pedaço 6	0 0 0	2 2 5	7 0 0	3 0 0	0 4 4	0 0 0
pedaço 7	0 1 1	1 1 3	1 3 3	0 0 0	0 0 0	3 2 3
pedaço 8	4 6 5	4 2 7	1 3 7	0 0 6	2 7 6	0 5 2
pedaço 9	0 2 6	4 4 4	0 0 0	4 6 2	4 2 0	0 0 0
	gene 1	**gene 2**	**gene 3**	**gene 4**	**gene 5**	**gene 6**
	dilatação		**união**	**união**	**união**	**união**

A Tabela 4.22 mostra a média dos tempos (em segundos) de todas as execuções nas linguagens escolhidas para desenvolvimento do algoritmo de modo sequencial de acordo com as quantidades de cromossomos escolhidas (n_crom). Em seguida, a Figura 4.40 mostra o gráfico da Tabela 4.22.

Tabela 4.22 - Telefone (Sequencial).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab	Octave
100	0,8620	8,0286	11,0020	12,8028	28,1634
200	1,2720	10,8372	13,7695	16,5861	54,0170
400	3,8000	44,5332	24,0483	25,5464	80,7294

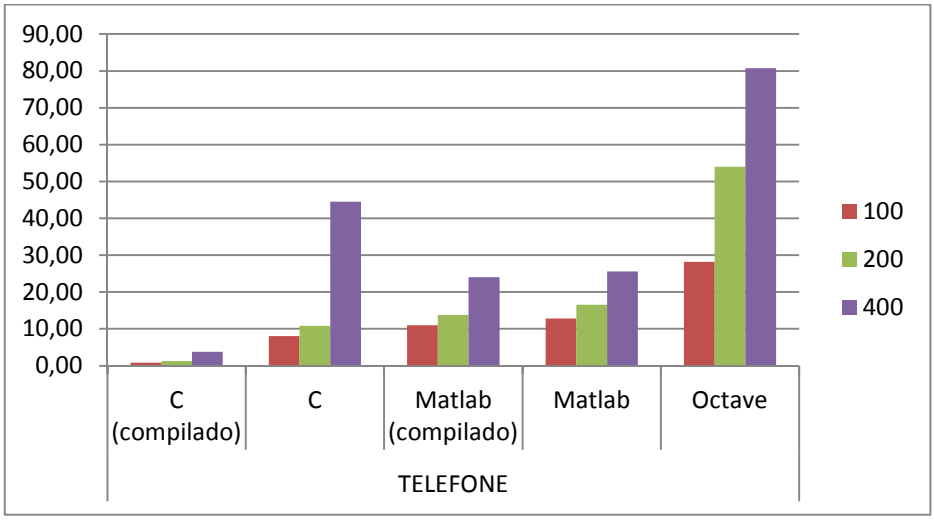

Figura 4.40 - Telefone (Sequencial).

Da mesma forma, a Tabela 4.23 e a Figura 4.41 retratam os resultados para o modo paralelo.

Tabela 4.23 - Telefone (Paralelo).

n_crom	C (compilado)	C	Matlab (compilado)	Matlab
100	0,2800	2,2200	7,7395	5,4326
200	0,9100	3,6740	10,0921	5,6924
400	1,2000	8,5780	9,9236	7,9690

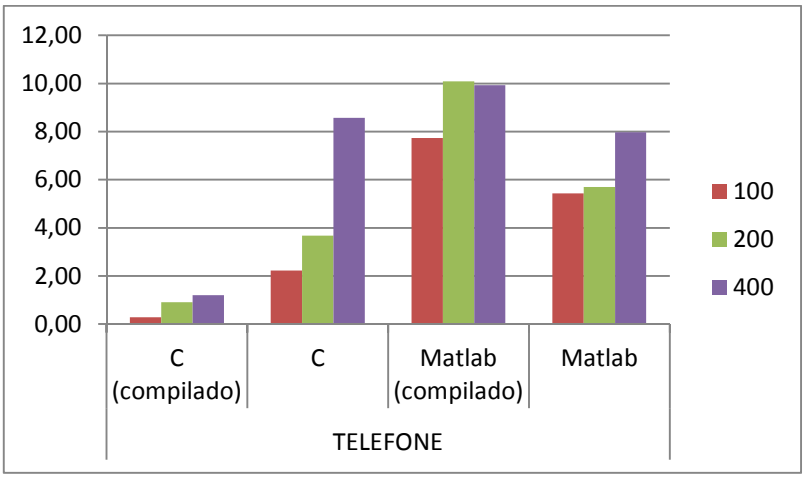

Figura 4.41 - Telefone (Paralelo).

4.1.5 Exemplo de Execução

Este exemplo de execução é o resultado do algoritmo decompondo a imagem do objeto caminhão utilizando linguagem Matlab sem a compilação do algoritmo e com execução sequencial.

Para este teste foram utilizadas as mesmas variáveis da Tabela 4.1 e tamanho da população de cromossomos igual a 100. As Figuras 4.42 à 4.50 ilustram os genes dos melhores cromossomos obtidos para cada pedaço da decomposição do objeto caminhão, bem como as dilatações entre os 1º e 2º genes e as uniões dos genes 3 a 6 resultando na subimagem completa compara a original.

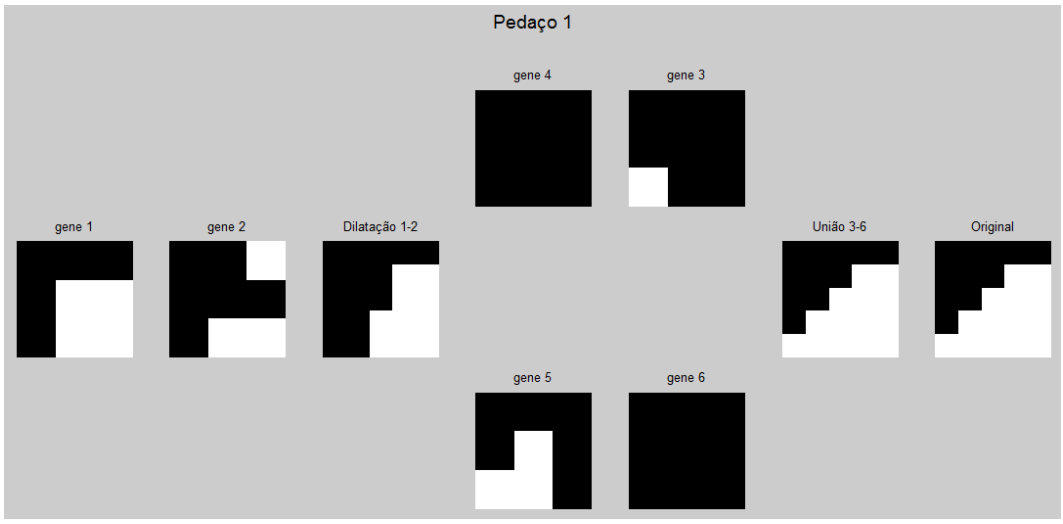

Figura 4.42 - Pedaço 1.

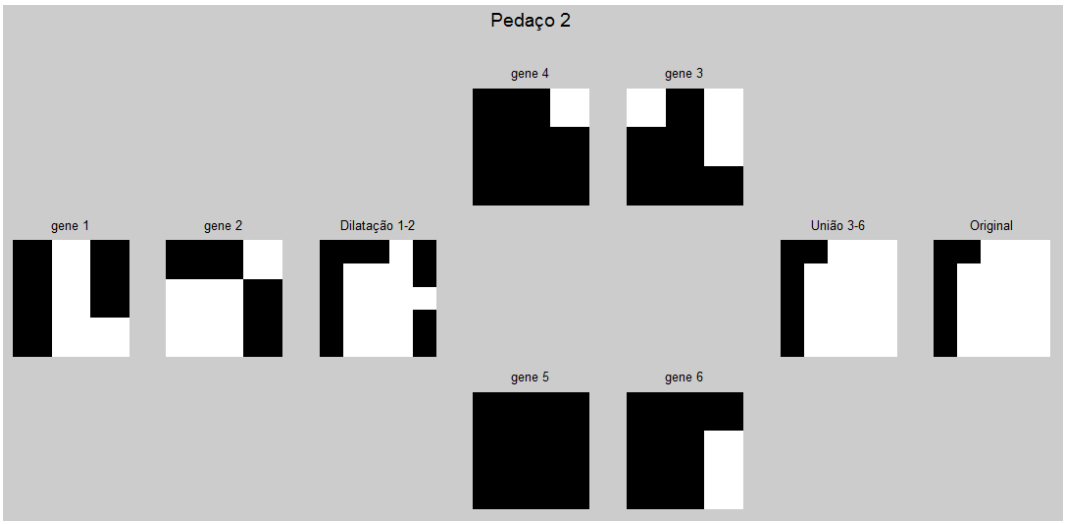

Figura 4.43 - Pedaço 2.

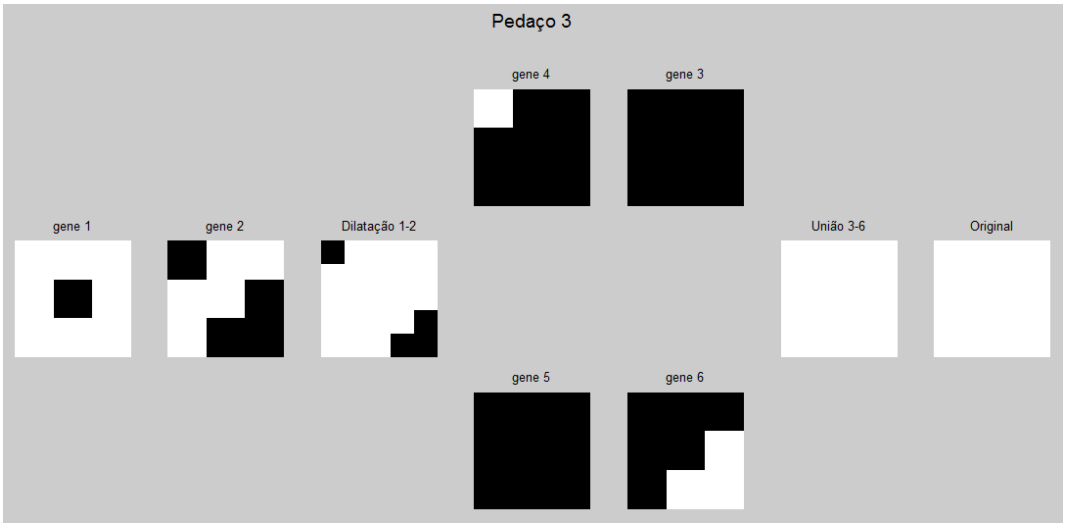

Figura 4.44 - Pedaço 3.

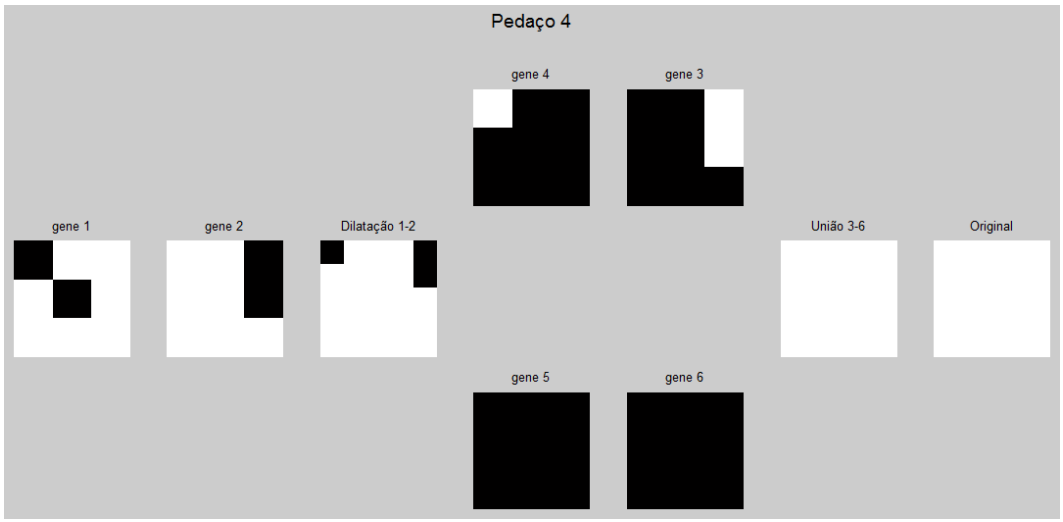

Figura 4.45 - Pedaço 4.

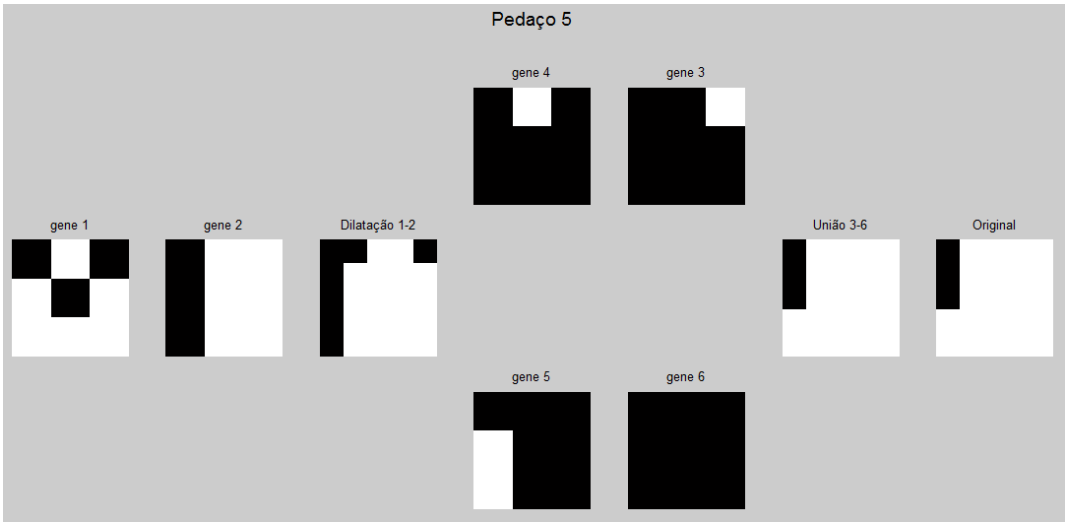

Figura 4.46 - Pedaço 5.

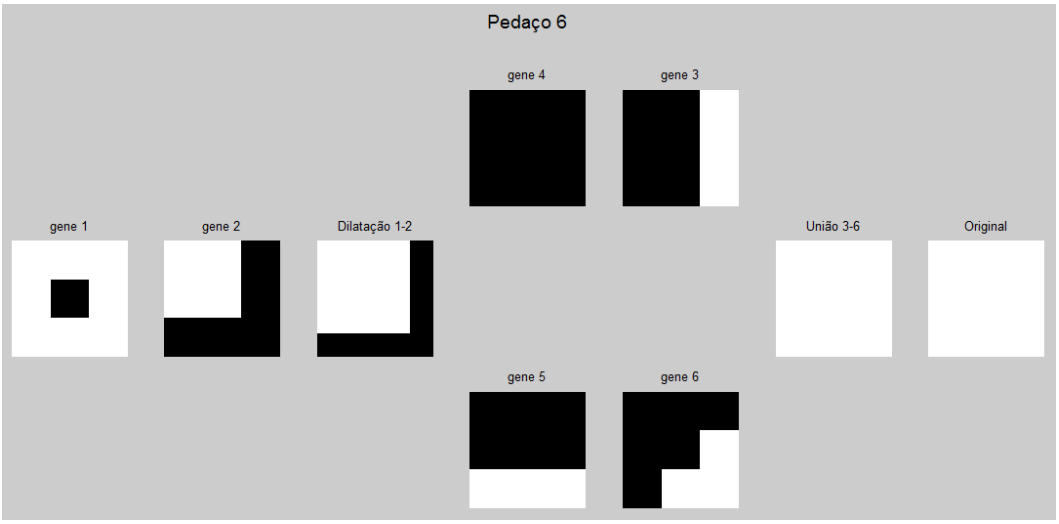

Figura 4.47 - Pedaço 6.

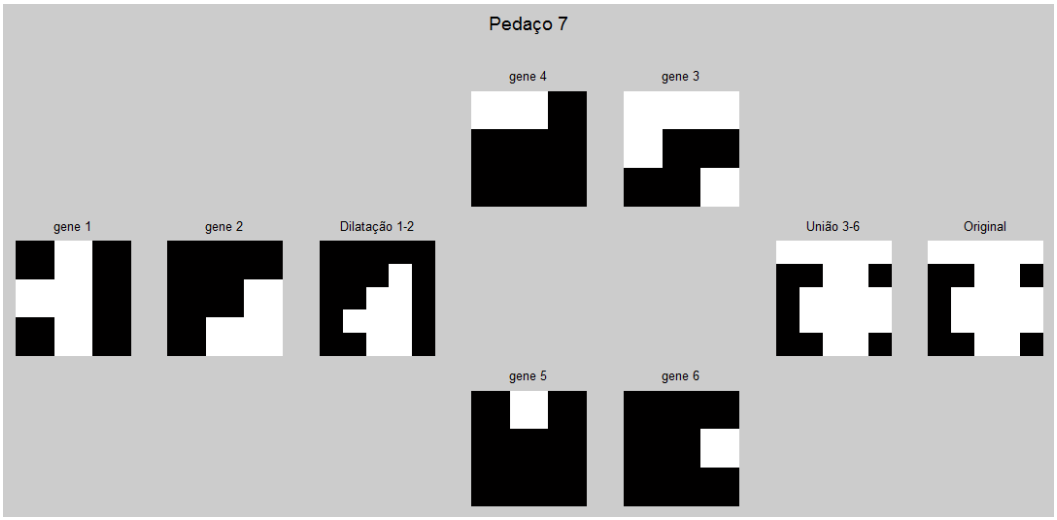

Figura 4.48 - Pedaço 7.

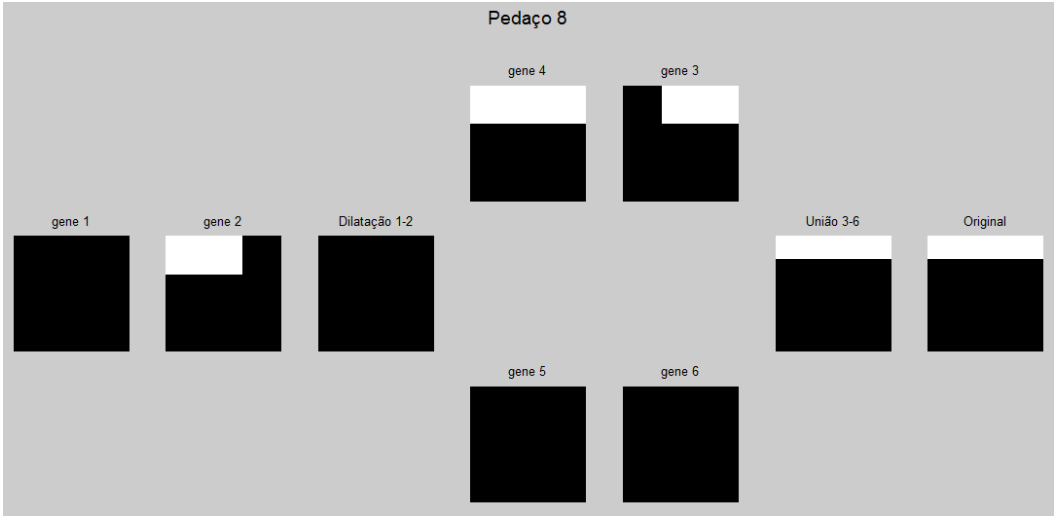

Figura 4.49 - Pedaço 8.

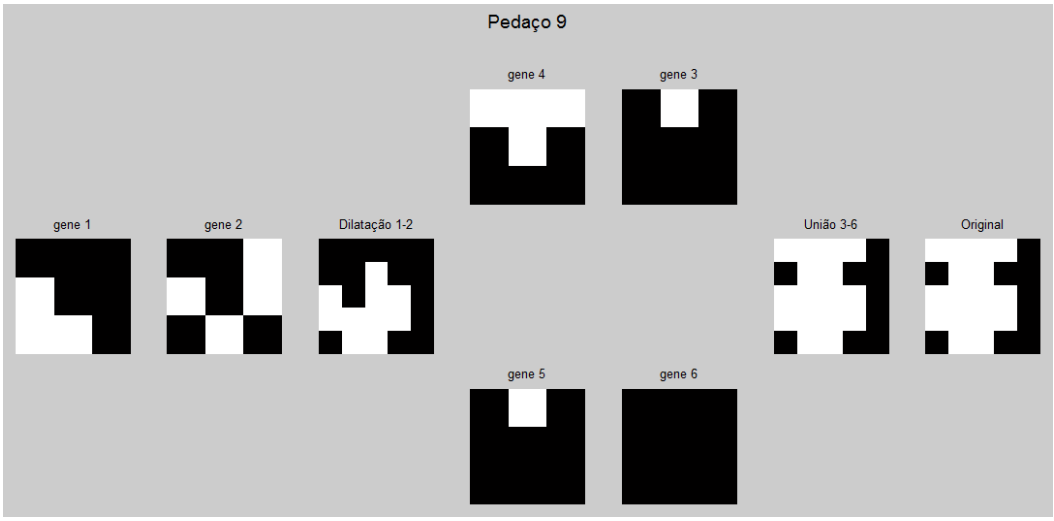

Figura 4.50 - Pedaço 9.

Ao final do processo de decomposição as Figuras 4.51 à 4.59 ilustram graficamente o número de gerações utilizadas para cada subimagem com o erro obtido pela diferença da imagem criada pela decomposição e a original. Após o encontro de erro igual a zero o algoritmo segue para as demais subimagens.

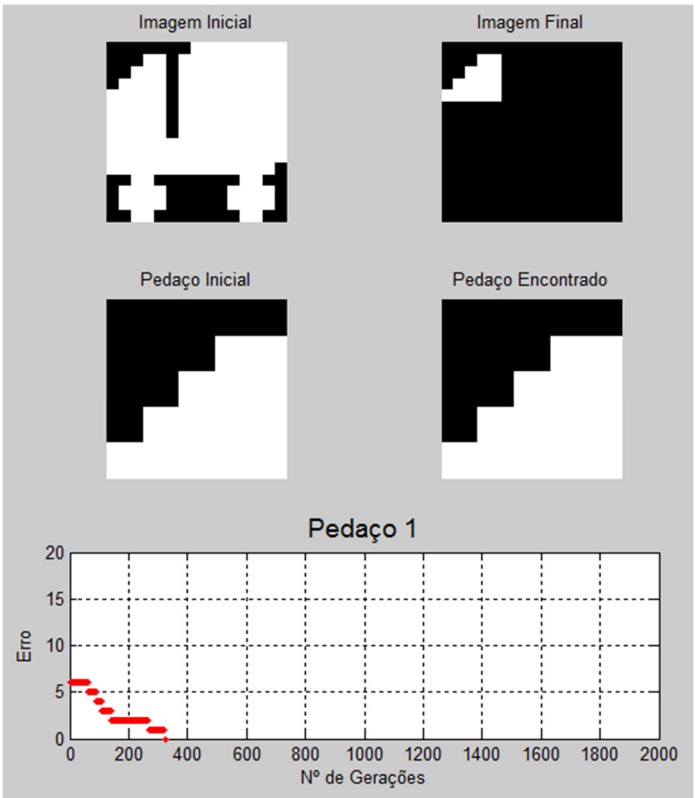

Figura 4.51 - Pedaço 1.

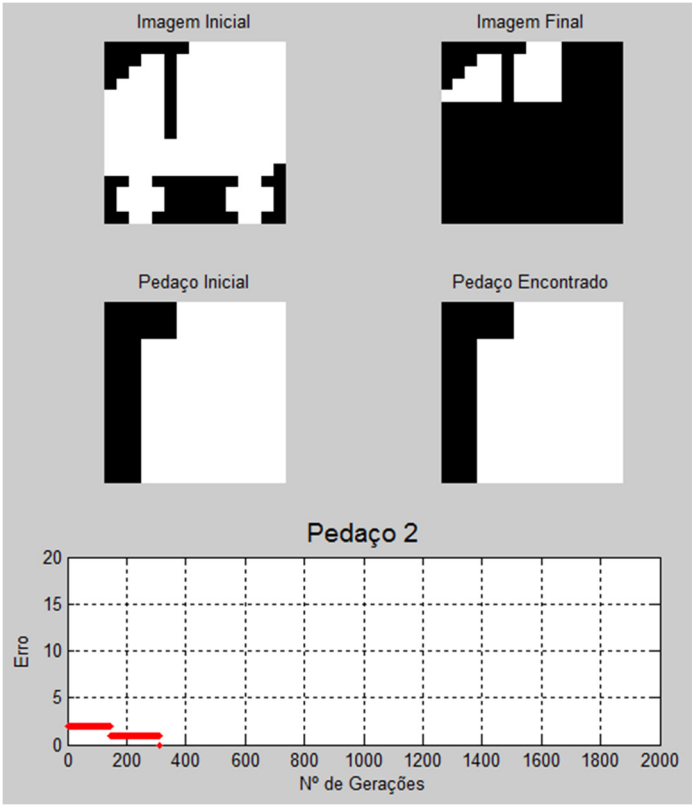

Figura 4.52 - Pedaço 2.

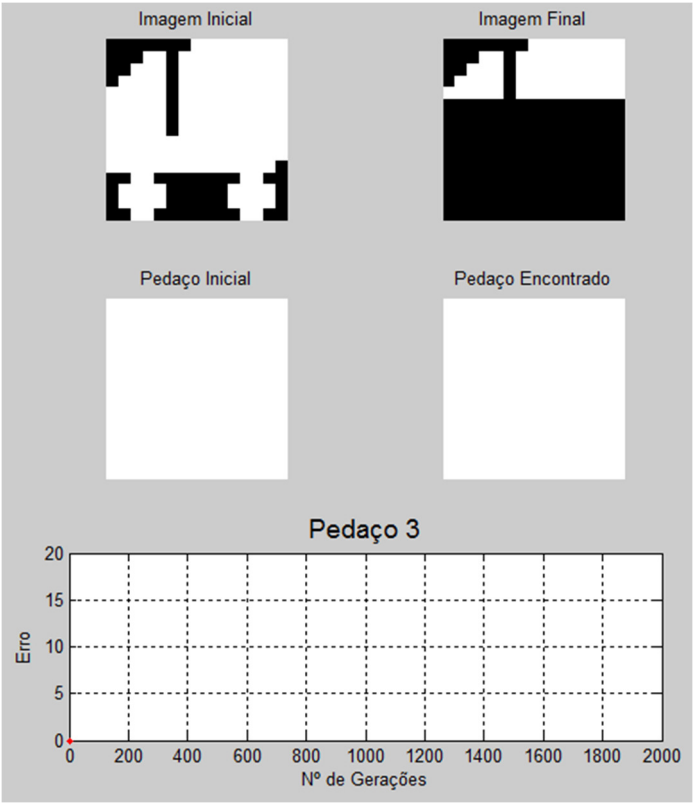

Figura 4.53 - Pedaço 3.

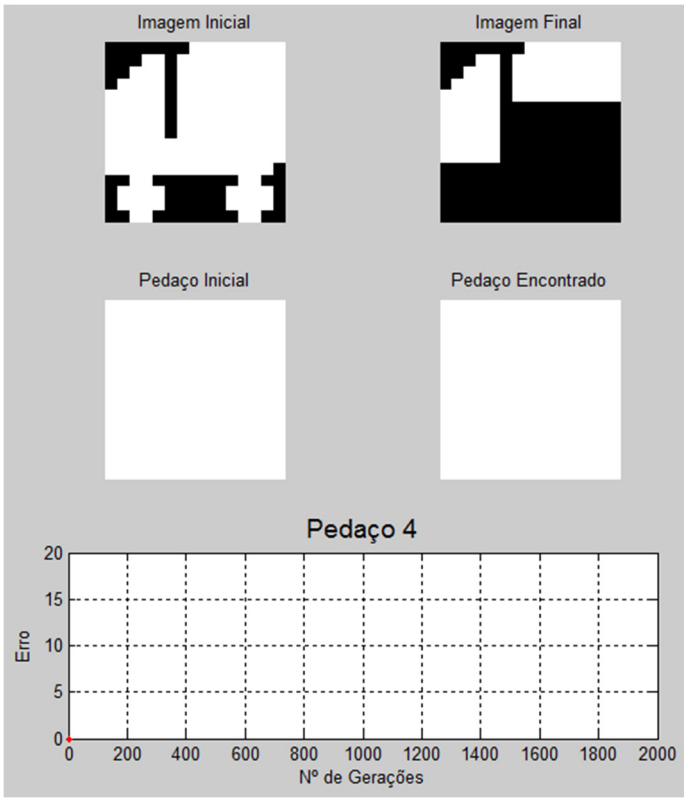

Figura 4.54 - Pedaço 4.

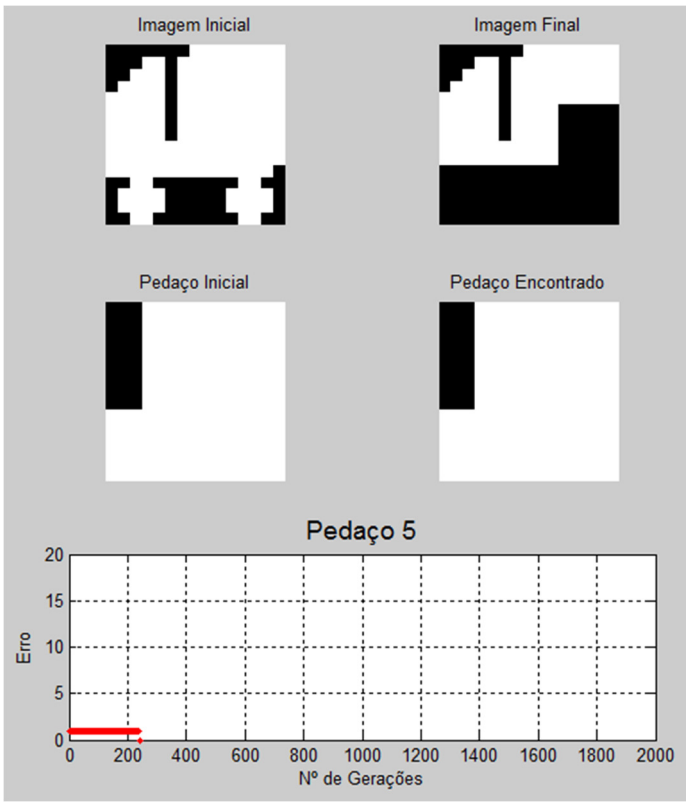

Figura 4.55 - Pedaço 5.

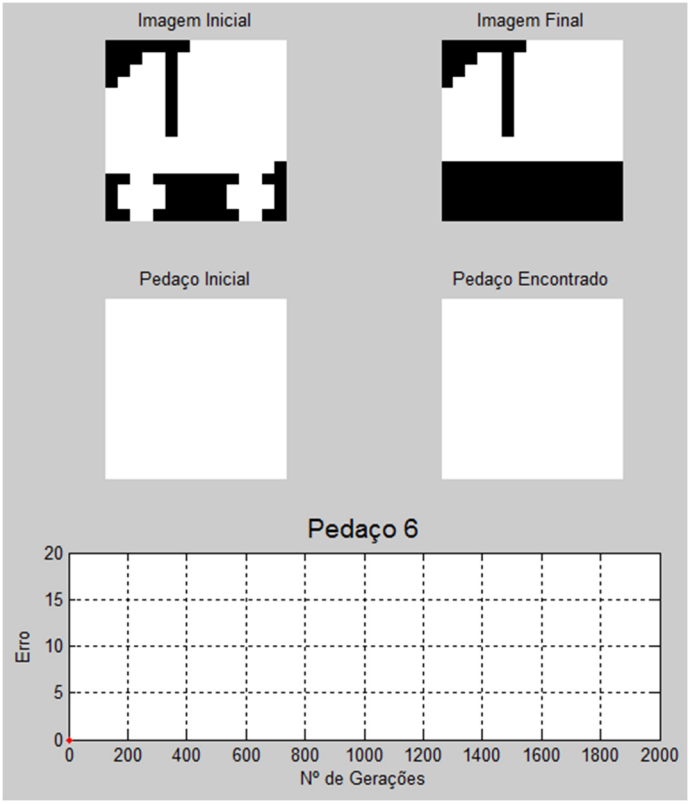

Figura 4.56 - Pedaço 6.

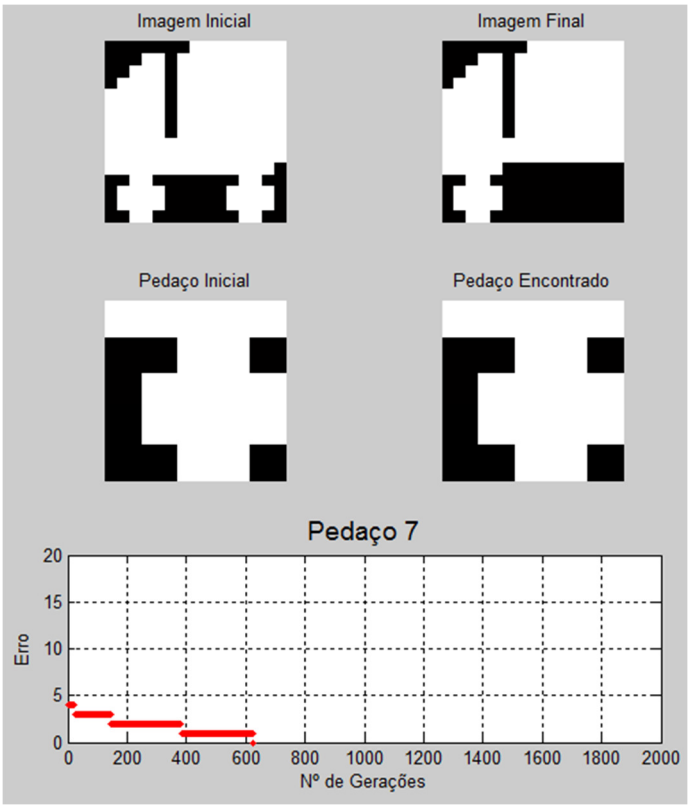

Figura 4.57 - Pedaço 7.

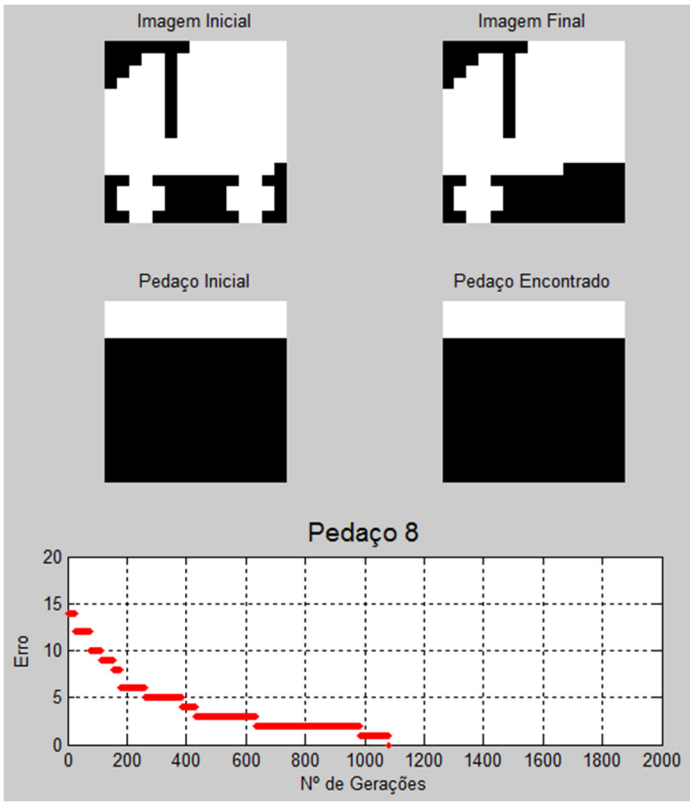

Figura 4.58 - Pedaço 8.

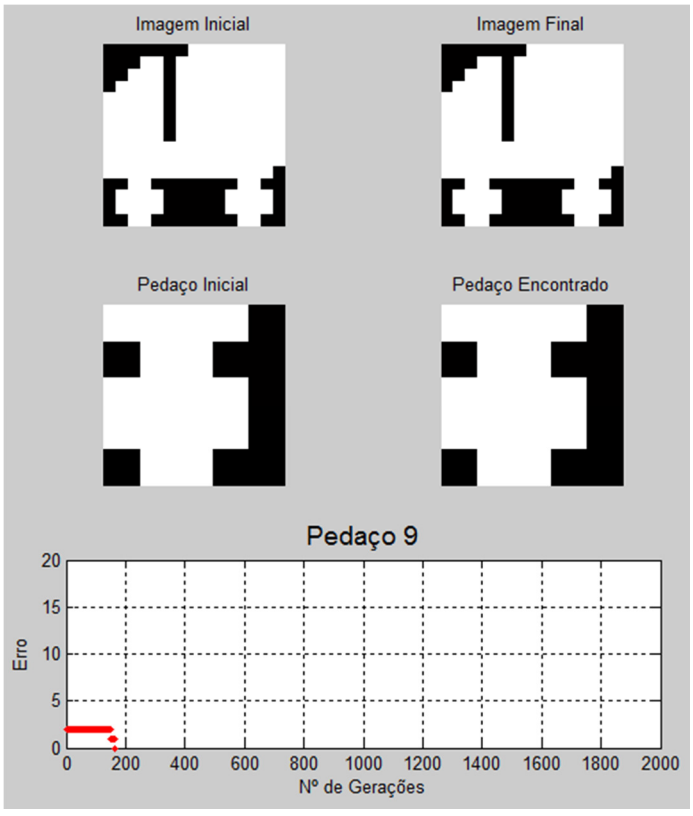

Figura 4.59 - Pedaço 9.

Após a execução do algoritmo, os dados referentes aos cromossomos de cada subimagem e o número de gerações utilizadas formam a Tabela 4.18.

Tabela 4.24 - Resultados do teste com caminhão e c = 100.

CROMOSSOMOS																	
0	3	3	1	0	3	0	0	4	0	0	0	0	2	6	0	0	0
2	2	3	1	6	6	5	1	0	1	0	0	0	0	0	0	1	1
7	5	7	3	6	4	0	0	0	4	0	0	0	0	0	0	1	3
3	5	7	6	6	7	1	1	0	4	0	0	0	0	0	0	0	0
2	5	7	3	3	3	1	0	0	2	0	0	0	4	4	0	0	0
7	5	7	6	6	0	1	1	1	0	0	0	0	0	7	0	1	3
2	6	2	0	1	3	7	4	1	6	0	0	2	0	0	0	1	0
0	0	0	6	0	0	3	0	0	7	0	0	0	0	0	0	0	0
0	4	6	1	5	2	2	0	0	7	2	0	2	0	0	0	0	0

4.1.6 Shih e Wu x Pillon

Shih e Wu (SHIH; WU, 2005) apresentam uma técnica baseada em algoritmos genéticos para se decompor eficientemente elementos estruturantes binários (formato arbitrário). Esta técnica é caracterizada por utilizar uma população inicial com base em 13 fatores primos, funções de fitness, adaptação do limiar dinamicamente e estratégia recursiva de redução de tamanho como recursos para melhorar o desempenho da decomposição, e é adequada para implementação paralela.

Segundo Shih e Wu (SHIH; WU, 2005), neste trabalho, apenas os elementos estruturantes de tamanho ímpar são mostrados. Entretanto, elementos estruturantes de tamanho par podem ser alterados para ímpar pela adição de uma coluna e linha adicional.

A partir desta técnica o tempo de execução foi menor e foram agregadas vantagens no custo computacional em comparação com Anelli (ANELLI et. al.,1998) e outros algoritmos genéticos. Já as comparações com outras decomposições que não utilizaram algoritmos genéticos, o algoritmo não demonstrou necessidade de numerosas regras de verificação.

Na Tabela 4.25 são mostradas as comparações entre a quantidade de *pixels* ativos (“1”) e uniões na imagem original a ser decomposta e a quantidade após a decomposição para todos os casos mostrados na Figura 4.60.

Tabela 4.25 - Comparações de custo dos elementos estruturantes originais e decompostos.

	Pato	Barco	Carro	Peixe	Abajur	Telefone
Pontos Originais	102	125	168	112	115	127
Uniões Originais	0	0	0	0	0	0
Pontos da Decomposição	33	47	46	33	35	42
Uniões da Decomposição	9	13	9	7	8	12

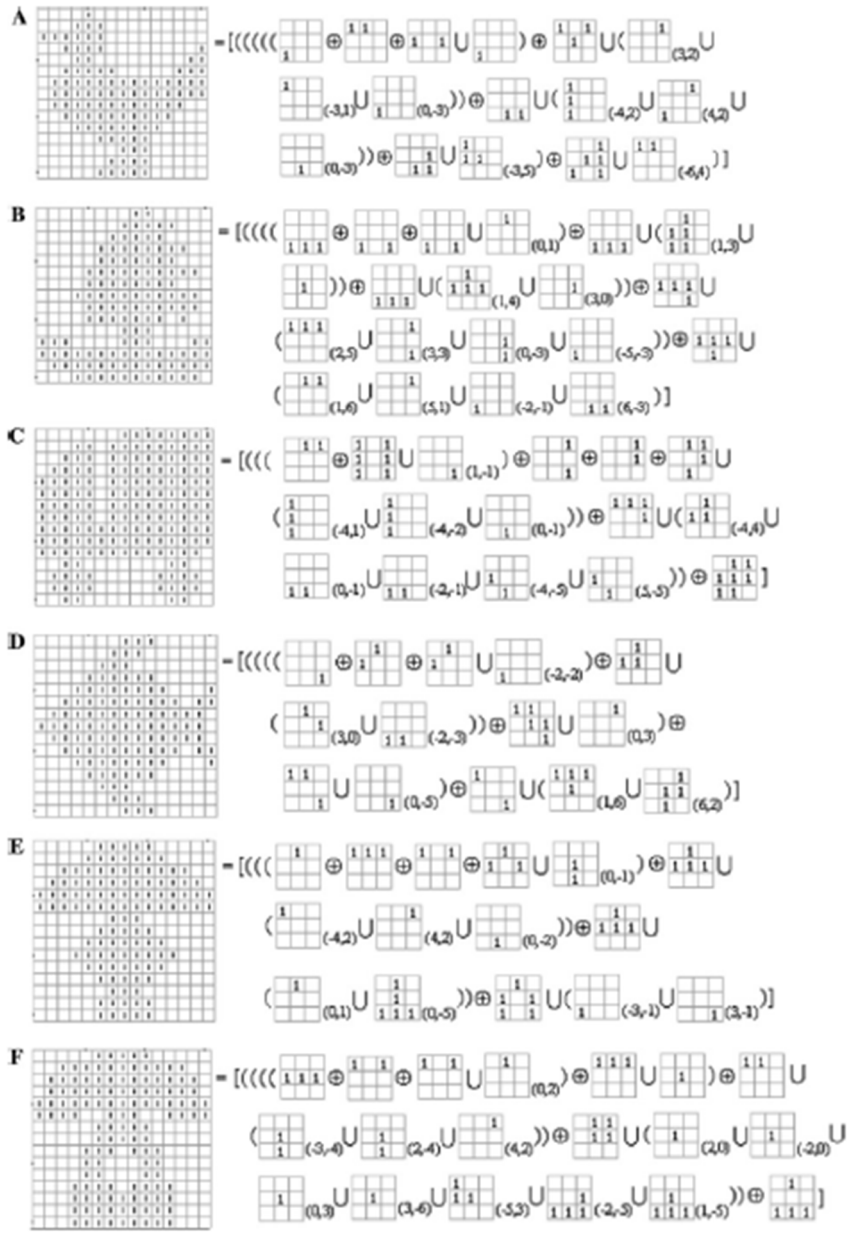

Figura 4.60 - Exemplos de diferentes formatos de imagem. (A) Pato, (B) Barco, (C) Carro, (D) Peixe, (E) Abajur e (F) Telefone. Fonte: Adaptado de (SHIH; WU, 2005).

Da mesma forma que Shih e Wu (SHIH; WU, 2005), este trabalho também apresenta uma técnica usando algoritmos genéticos para decompor eficientemente elementos estruturantes binários em formato arbitrário.

Para a simplificação do processo de decomposição, adotou-se uma padronização referente ao comprimento do cromossomo e a quantidade das operações executadas nos elementos estruturantes, totalizando assim, com as nove subimagens de uma elemento 15x15, o cromossomo terá 9 dilatações (1 para cada subimagem) e 36 uniões (4 para cada subimagem) em comparação ao proposto por Shih e Wu (SHIH; WU, 2005) que possui 6 dilatações (para dilatar um elemento 3x3 até 15x15 são necessários 6 dilatações) e “n” uniões (o número de uniões depende do formato do elemento).

Esta simplificação (Figura 4.61) alcançou menos tempo em comparação com os trabalhos de Anelli (ANELLI et. al., 1998) e Shih (SHIH; WU, 2005). Anelli utilizando dois processadores HP9000 com 128 megabytes de memória RAM precisou de cerca de seis horas para decompor um elemento de tamanho 16x16 e Shih (SHIH; WU, 2005) utilizando um processador Pentium III/866MHz com 256 megabytes de memória RAM levou cerca de 20 minutos para um elemento 17x17. A otimização adotada utilizando um Corei7/3.4GHz e 8 gigabytes de memória RAM decompôs em cerca de 0,236 segundos (execução sequencial) e 0,102 segundos (execução paralela) um elemento 15x15. Vale ressaltar, que para ser justa a comparação acima citada, seria necessário a utilização do mesmo computador para executar cada algoritmo.

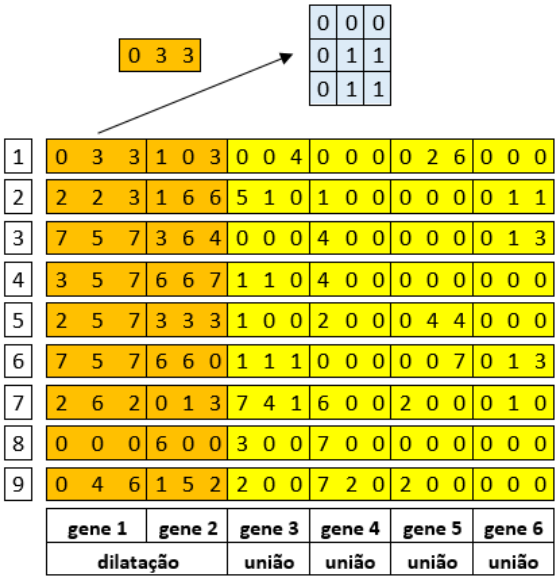

Figura 4.61 – Estrutura do Cromossomo.

Portanto, a estratégia adotada torna viável a implementação em *hardware* (onde muitas vezes é limitado), visto que, com uma estrutura simples e fixa para alocação dos elementos e execução das operações de dilatação e união ao contrário do proposto por Shih e Wu (SHIH; WU, 2005) obteve ótimos resultados tanto com a execução sequencial quanto na paralela, utilizando o processamento máximo (todos os núcleos do processador).

Capítulo 5

CONCLUSÃO

Este capítulo de conclusão é a parte final do trabalho na qual é apresentada uma síntese dos principais resultados e da proposta para solução do problema estabelecido inicialmente. Este capítulo está dividido em três seções: 5.1 Considerações Finais; 5.2 Contribuições e Limitações; 5.3 Trabalhos Futuros.

5.1 Considerações Finais

Este trabalho teve por objetivo comparar metodologias para a decomposição eficiente dos elementos estruturantes para morfologia matemática binária visando uma simplificação do processo elaborado por Shih (SHIH; WU, 2005) descrito em seu artigo. O algoritmo foi implementado utilizando as ferramentas Visual Studio 2012, Matlab 2013a e Octave, executando o método de duas formas: sequencial e paralelo.

Neste trabalho foram expostos os resultados obtidos com o algoritmo desenvolvido para os estudos de caso (seis imagens) relacionados ao artigo de Shih (SHIH; WU, 2005) demostrando a viabilidade de uma futura implementação em *hardware* através da padronização na estrutura de decomposição e seus operadores de forma simples e objetiva.

5.2 Contribuições e Limitações

O objetivo deste trabalho foi a elaboração de um algoritmo que fosse capaz de simplificar a decomposição de imagens viabilizando sua implementação em *hardware*. Com auxílio dos métodos de decomposição existentes e o uso de algoritmos genéticos para evolução da decomposição na busca do cromossomo ideal, a implementação do algoritmo desenvolvido proporcionou uma estrutura simples, compacta e flexível para ser usada no processamento de imagens.

Os testes do algoritmo desenvolvido foram conduzidos utilizando variações no tamanho da população de cromossomos, modo de execução do algoritmo (sequencial e paralelo) e linguagem de programação no intuito de obter uma base para implementação em *hardware*.

Após os testes realizados, o algoritmo compilado na linguagem C no modo paralelo obteve o melhor resultado para o tempo de decomposição com uma população de 100 indivíduos. Os demais tamanhos (200 e 400) também obtiveram êxito na decomposição porém proporcionalmente ao tamanho, o tempo também foi maior. No caso das linguagens de programação, o uso do *software* Matlab foi essencial para as simulações feitas durante a fase inicial do desenvolvimento do algoritmo pela sua gama de funcionalidades prontas ser muito maior que a linguagem C, ao qual teve que se desenvolver algumas funções já presentes no Matlab. Com o auxílio do Matlab todos os problemas ocasionados durante a criação do algoritmo puderam ser resolvidos, reduzindo assim o tempo de execução das primeiras vezes (cerca de horas) para menos de 1 segundo em C.

Portanto, é possível afirmar que o uso de todas as ferramentas disponíveis descritas neste trabalho fora bem sucedidas e colaborou para o desenvolvimento da simplificação e otimização da estrutura da decomposição de elementos estruturantes em morfologia matemática binária permitindo sua implementação em *hardware* com programação paralela.

5.3 Trabalhos Futuros

Como trabalhos futuros, pretende-se realizar: uso de GPU's para acelerar ainda mais o tempo de execução do algoritmo em *software* e implementação em *hardware* para estudar uma comparação entre todos os métodos desenvolvidos.

REFERÊNCIAS

ANELLI, G.; BROGGI, A.; DESTRI, G. Decomposition of arbitrarily shaped binary morphological structuring elements using genetic algorithms, IEEE Trans. Pattern Anal. Mach. Intell. 20 (2) p. 217–224, 1998.

FACON, J. Morfologia Matemática: Teorias e Exemplos. Editora Universitária Champagnat da Pontifícia Universidade Católica do Paraná. Curitiba, 320 p., 1996.

FREEMAN, H. Computer processing of line-drawing images. Comput. Surveys, vol. 6, no. 1, p. 57-97, Mar. 1974.

GOLDBERG, D. E. Genetic Algorithms in Search Optimization and Machine Learning, Addison-Wesley, 412 p., 1989.

GONZALEZ, Rafael C.; WOODS, Richard E. Digital Image Processing. Reading, MA: Addison-Wesley, 716 p. 1992.

HOLLAND, J. H. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. MIT Press. 2nd Edition. 1992.

KOZA, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. [S. l.]: MIT Press, 813 p., 1992.

LINDEN, R. Algoritmos Genéticos - Uma importante ferramenta da inteligência computacional - 2ª Edição. BR: Brasport, 2008.

MARQUES FILHO, O. & VIEIRA NETO, H. Processamento digital de Imagens. Rio de Janeiro, Brasport, 1999.

MITCHELL, M. An Introduction To Genetic Algorithms. MIT Press. 158 p., 1996.

OBITKO, M.; SLAVÍK, P. Visualization of Genetic Algorithms in a Learning Environment. Brastisiana: Comenius University, 1999.

PARK, H.; CHIN, R.T. Optimal decomposition of convex morphological structuring elements for 4-connected parallel array processors. IEEE Trans. Pattern Anal. Machine Intell. Vol. 16, No. 3, p. 304-313, Mar. 1994.

PARK, H.; CHIN, R.T. Decomposition of arbitrarily shaped morphological structuring elements, IEEE Trans. Pattern Anal. Mach. Intell. p. 2-15, 1995.

PARK, H.; YOO, J. Structuring element decomposition for efficient implementation of morphological filters. IEEE Proceedings of Vision, Image and Signal Processing. p. 31-35, 2001.

PARKER, J. R. Algorithms for Image Processing and Computer Vision. New York, John wiley & Sons, 1997.

PEDRINO, E. C. Arquitetura pipeline reconfigurável através de instruções geradas por programação genética para processamento morfológico de imagens digitais utilizando FPGAs. 2008. 220 p. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

REZENDE, S. O. ET ALLI. Sistemas Baseados em Conhecimento. In: Sistemas Inteligentes: fundamentos e aplicações. Editora Manole. Barueri, SP. 2003.

RICHARDSON, C. H.; SCHAFER, R. W.A Lower Bound for Structuring Element Decomposition. IEEE Trans. No.4, 365-369, April 1991.

RUSS, John C. The image processing handbook.2 ed., Boca Raton, CRC Press, 1995.

SHIH, F. Y.; WU Y. Decomposition of binary morphological structuring elements based on genetic algorithms. Comput. Vis. Image Underst. 99, 2, p. 291-302, August 2005.

SOILLE, P. Morphological Image Analysis. Principles and Applications. Berlin: Springer-Verlag, 316 p., 1999.

WANGENHEIM, A. Seminário Introdução à Visão Computacional. Disciplina do Curso de Pós-Graduação em Ciência da Computação. Florianópolis: Universidade Federal de Santa Catarina. Disponível em: <http://www.inf.ufsc.br/~visao/>. Acessado em: 31/08/2012.

WEN-YANG l.; WEN-YUNG l.; TZUNG-PEI H. Adapting Crossover and Mutation Rates in Genetic Algorithms. J. Inf. Sci Eng., 2003.

XU, J. Decomposition of convex polygonal morphological structuring elements into neighborhood subsets. IEEE Trans. Pattern Anal. Machine Intell. Vol. 13, No. 2, p. 153-162, Feb. 1991.

ZHUANG, X.; HARALICK, R. M. Morphological structuring element decomposition, Comput. Vision Graph. Image Process. p. 370-382, 1986.