# DISSERTAÇÃO DE MESTRADO


**UNIVERSIDADE FEDERAL DE SÃO CARLOS**

**CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA**

**PROGRAMA DE PÓS-GRADUAÇÃO EM**

**CIÊNCIA DA COMPUTAÇÃO**


## "Simulador de Arquitetura para Processamento de Imagens Usando Programação Genética Cartesiana"


**ALUNO: Paulo Cesar Donizeti Paris**

**ORIENTADOR:** Prof. Dr. Emerson Carlos Pedrino


**São Carlos**
**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

# SIMULADOR DE ARQUITETURA PARA PROCESSAMENTO DE IMAGENS USANDO PROGRAMAÇÃO GENÉTICA CARTESIANA

**PAULO CESAR DONIZETI PARIS**

**ORIENTADOR: PROF. DR. EMERSON CARLOS PEDRINO**

São Carlos - SP
2014

**UNIVERSIDADE FEDERAL DE SÃO CARLOS**

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# SIMULADOR DE ARQUITETURA PARA PROCESSAMENTO DE IMAGENS USANDO PROGRAMAÇÃO GENÉTICA CARTESIANA

**PAULO CESAR DONIZETI PARIS**

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: Arquitetura e Processamento de Imagens e Sinais.
Orientador: Prof. Dr. Emerson Carlos Pedrino

São Carlos - SP
2014

**Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar**

P232sa

Paris, Paulo Cesar Donizeti.
Simulador de arquitetura para processamento de imagens usando programação genética cartesiana / Paulo Cesar Donizeti Paris. -- São Carlos : UFSCar, 2014.
94 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2013.

1. Processamento de imagens. 2. Morfologia matemática. 3. Programação genética (Computação). 4. Operadores de imagem. 5. Procedimento genético. 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**

# "Simulador de Arquitetura para Processamento de Imagens Usando Programação Genética Cartesiana"

Paulo Cesar Donizeti Paris

**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:**

Emerson Carlos Pedrino

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

**Prof. Dr. José Hiroki Saito**
**(PPG-CC/ UFSCar)**

Adriane B. de S. Serapião

**Profa. Dra. Adriane Beatriz de Souza Serapião**
**(IGCE/UNESP)**

**São Carlos**
**Dezembro/2013**

# Dedicatória

Dedico este trabalho de mestrado à minha mãe Zulmira (in memoriam), ao meu inseparável pai Sebastião, à minha linda esposa Carolina, à minha filha Laura, que é um presente de Deus, à minha sogra Regina, ao meu sogro Heleno, às minhas irmãs Eva e Luciene, aos meus sobrinhos, à dona Madalena, ao meu amigo Hiroshi Tejima, e em especial ao meu orientador e amigo Emerson, sem o qual este trabalho não seria possível.

# AGRADECIMENTOS

Agradeço a Deus, por tudo que tem feito em minha vida: pela alegria de viver, por minha família, pelos meus amigos, pelo ar que respiro, pelos dons que me deu e pelos relacionamentos que possibilitam o meu crescimento tanto pessoal quanto profissional a cada dia. Agradeço a São Judas Tadeu, por interceder por mim diante de Deus. Agradeço à minha esposa Carolina, pela paciência e apoio em todos os momentos. À minha filha Laura pelos lindos sorrisos que me dão ainda mais força para continuar. Agradeço a toda a minha família, por estarem sempre ao meu lado. Agradeço ao meu orientador e grande amigo Emerson, pelos ensinamentos com tanta atenção e paciência. Agradeço à professora Maria do Carmo Nicoletti, pela especial contribuição neste trabalho. Agradeço ao Pillon, um amigo de todas as etapas deste trabalho. Agradeço aos professores Hiroshi Tejima, Ernesto Urquieta, Ildeberto Bugatti e Antônio Francisco do Prado, por me incentivarem desde o início de minha carreira. Agradeço aos professores Marcio Merino Fernandes e Edilson Reis Rodrigues Kato, por terem participado da minha banca de qualificação. Agradeço ao professor José Hiroki Saito e a professora Adriane Beatriz de Souza Serapião por terem participado da minha banca de defesa de dissertação. Agradeço aos meus amigos do departamento de Computação da UFSCAR. Agradeço aos meus amigos do CCET/UFSCar, Vera, Sergio, Claudia, professor Paulo Caetano e professora Sheyla, pelo apoio em muitas ocasiões. Agradeço à UFSCar, pelo afastamento parcial que me foi concedido durante este período,

# RESUMO

## SIMULADOR DE ARQUITETURA PARA PROCESSAMENTO DE IMAGENS USANDO PROGRAMAÇÃO GENÉTICA CARTESIANA

As ferramentas oferecidas pela área de Morfologia Matemática são muito eficientes quando aplicadas na análise de imagens binárias, o que é de grande importância em áreas como: visão robótica, inspeção visual, entre outras. Tais ferramentas aliadas à Computação Evolucionária e baseadas em mapeamentos genótipo-fenótipo permite que as tarefas computacionais possam ser executadas de forma automática, sem programações explicitas, o que leva a uma motivação na busca de uma forma de redução do grau de dificuldade, muitas vezes encontrado pelos especialistas na realização de tarefas de seleção de operadores de imagem para serem utilizados em tarefas de análise. Além disso, se tais tarefas necessitarem de processamentos rápidos sobre as imagens, faz-se necessário o uso de arquiteturas implementadas em *hardware*, o que também não é muito trivial de serem projetadas. Assim, neste trabalho, implementa-se um simulador de arquiteturas de *hardware* para processamento de imagens, com base na metodologia de Programação Genética Cartesiana, que gera automaticamente filtros para o processamento de imagens binárias, ou seja, constrói-se automaticamente uma sequência de operadores lógicos e morfológicos que produzem os filtros para as imagens desejadas. Os resultados obtidos a partir de diversos estudos de casos de transformação dessas imagens são apresentados e analisadoscomparativamente em relação aos resultados anteriores disponíveis na literatura. Com base nestes resultados, é possível estudar o comportamento de tal arquitetura, através da variação dos parâmetros do procedimento genético no ambiente do simulador. Assim, é possível inferir se a arquitetura modelada será ou não adequada à aplicação desejada, logo, facilitando-se o processo de projeto e implementação em *hardware*.

**Palavras-chave**: Morfologia Matemática, Programação Genética Cartesiana, Operadores de Imagem, Procedimento Genético.

# ABSTRACT


## Hardware Architecture Simulator for Image Processing Using Cartesian Genetic Programming

The tools offered by the area of Mathematical Morphology are very effective when applied to the analysis of binary images, which it is of great importance in areas such as: robotic vision, visual inspection, among others. Such tools, beside to Evolutionary Computation and based on genotype-phenotypes mappings allow computational tasks be performed automatically without explicit programming, which leads to the motivation, in the search of a way of reducing the degree of difficulty often found by human experts in performing tasks of selecting linear operators to be used in morphological filters. Moreover, if such tasks require fast processing on the images, it is necessary the use of architectures implemented in hardware, which it is not too trivial to be done. In this work, a hardware architecture simulator has been implemented for image processing, based on Cartesian Genetic Programming, which automatically builds filters for processing binary images, i.e., automatically build a sequence of logical and morphological operators that produces filters to obtain an approximate of the desired images. The results obtained from several experiments of transformation of these images are presented and comparatively analyzed in relation to previous results available in the literature. Based on these results, it will be possible to study the behavior of such architecture, through the variation of the parameters of the genetic procedure in the simulator environment. Thus, it will be possible to infer if the architecture is suitable or not for a desired application, so facilitating the process of design and implementation of it in hardware.



**Keywords**: Mathematical Morphology, Cartesian Genetic Programming, Image Operators, Genetic Procedure.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

**AI** – Artificial Intelligence

**CGP** – Cartesian Genetic Programming

**CGP-IP** – Cartesian Genetic Programming for Image Processing Tasks

**CLB** – *Configuration Logical Blocks*

**DAG** – *Directed Acyclic Graph*

**EA** – Evolutionary Algorithm

**EC** – Evolutionary Computation

**EHW** – Intrinsic Evolvable Hardware

**EP** – Evolutionary Programming

**ES** – Strategy Evolutionary

**FE** – Function Element

**FPGA** – Field Programmable Gate Array

**GA** – Genetic Algorithm

**GP** – Genetic Programming

**HDL** – *Hardware Description Language*

**LUT** – *Look-up Table*

**MAE** – *Mean Absolute Error*

**MER-A** – *Mars Exploration Rover*

**MCC** – *Matthews Correlation Coefficient*

**MM** – Mathematical Morphology

**MNIST** – Mixed National Institute of Standards and Technology Database

**OpenCV** – Open Source Computer Vision Library

**SE** – Structuring Element

**VRA** – Virtual Reconfigurable Architecture

**SHAIP-CGP** – Simulador de Arquitetura para Processamento de Imagens Usando Programação Genética Cartesiana

# SUMÁRIO

# Capítulo 1

# *INTRODUÇÃO*

*Neste capítulo estão descritos o contexto em que este trabalho está inserido e a motivação que deu origem a este projeto de pesquisa. Em seguida são discutidos os objetivos a serem alcançados, finalizando com a descrição da organização desta dissertação. Para manter a consistência de nomenclatura com a literatura da área de estudos, optou-se neste trabalho por manter as siglas utilizadas como originalmente foram sugeridas na literatura.*

## 1.1 Contexto

O propósito deste trabalho de dissertação de mestrado é desenvolver um sistema automático para a construção de filtros para processamento de imagens binárias. Este sistema é baseado numa abordagem conhecida como Programação Genética Cartesiana (*CGP - Cartesian Genetic Programming*), que é uma das ramificações da computação evolucionária (*EC - Evolutionary Computation*). O sistema foi desenvolvido, como um simulador de arquitetura de *hardware* evolutivo e nomeado (SHAIP-CGP - *Hardware Architecture Simulator for Image Processing Using Cartesian Genetic Programming*), na qual toda a implementação foi desenvolvida em ambiente de programação com o *software Matlab (versão educacional)*.

## 1.2 Motivação e Objetivos

Resolver problemas relacionados a processamento de imagens digitais, bem como de visão computacional, geralmente requer, de antemão, uma análise detalhada e cuidadosa do problema, comumente realizada manualmente por profissionais experientes. Geralmente soluções para estes tipos de problemas, dependem da seleção adequada do conjunto de operadores de imagem, bem como encontrar sua sequência apropriada. Quando estas tarefas são executadas por seres humanos, todo o processo é muito lento, devido ao julgamento e erro comumente usados.

A natureza *ad-hoc* do problema, invariavelmente transforma a busca pela melhor sequência de operadores de imagem em uma tarefa complexa e sua solução inadequada para reutilização. Muitas pesquisas disponíveis em trabalhos publicados ao longo dos últimos anos têm focado seus esforços tentando reduzir a complexidade envolvida dentro da concepção de novos operadores de imagens que sejam capazes de executar várias tarefas computacionais.

Particularmente, ferramentas computacionais baseadas no formalismo da morfologia matemática (MM – *Mathematical Morphology*) são consideradas eficazes quando sua abordagem é aplicada em problemas práticos e teóricos de diversas áreas, tais como: análise e processamento de imagens. Desenvolvimentos nessas áreas têm grande importância e impacto em áreas subjacentes, como visão robótica, inspeção visual, medicina, análise de texturas, entre outras.

Portanto, o objetivo deste trabalho é simular uma arquitetura em Programação Genética Cartesiana (*CGP - Cartesian Genetic Programming*) para estudar o funcionamento de uma possível implementação em *hardware* para processamento de imagens binárias no futuro. Com base nos resultados a serem encontrados, será possível estudar o comportamento de tal arquitetura através da variação dos parâmetros do procedimento genético no ambiente do simulador. Assim, será possível inferir se a arquitetura modelada será ou não

adequada à aplicação desejada, logo, facilitando seu processo de projeto em *hardwar*e.

## 1.3 Organização do Trabalho

Esta dissertação de mestrado possui uma estrutura que se divide em seis capítulos apresentando os seguintes conteúdos:

- O Capítulo 1, denominado Introdução está dividido em três seções: 1.1, Contexto; 1.2, Motivação e Objetivos; 1.3, Organização do Trabalho.
- O Capítulo 2, denominado de fundamentação teórica, apresenta as fundamentos teóricos necessários para a execução do projeto em si; sua divisão possui as seguintes seções: 2.1, Morfologia Matemática; 2.2, Conceitos Básicos da Computação Evolucionária; 2.3, Considerações Finais.
- O Capítulo 3 é chamado de Revisão Relacionada e está dividido em seis seções: 3.1, CGP para processamento de imagens; 3.2, Classificações de terreno em Marte, usando CGP-IP; 3.3, CGP utilizado para tarefas de processamento de Imagens; 3.4, CGP para reconhecimento de caracteres manuscritos; 3.5, Arquitetura reconfigurável virtual para *hardware* evolutivo (artigo base); 3.6, Sistema baseado em GP para a construção automática de filtros de imagem (comparação dos resultados).
- O Capítulo 4, denominado Proposta do Trabalho, possui uma seção dividida em sete subseções abordando o desenvolvimento do simulador de arquitetura: 4.1, Simulador de *hardware* para processamento de imagem baseado numa abordagem da computação evolucionária.
- O capítulo 6 de Conclusão está dividido em quatro seções: 6.1, Considerações Finais; 6.2, Contribuições e Limitações; Lições Aprendidas; Trabalhos Futuros.
- Por fim, as Referências Bibliográficas.

# Capítulo 2

# *FUNDAMENTAÇÃO TEÓRICA*

*Neste capítulo são descritos os conceitos e os fundamentos teóricos (baseados na literatura), necessários para a compreensão da proposta de se desenvolver um simulador de arquitetura (software), capaz de construir automaticamente filtros morfológicos para imagens binárias. Assim, na seção 2.1, são apresentados os conceitos relacionados à Morfologia Matemática. Na seção 2.2 são descritos os conceitos referentes à Computação Evolucionária. Por fim, na seção 2.3, são apresentadas as Considerações Finais.*

## 2.1 Morfologia Matemática

Morfologia Matemática (MM) é uma teoria que fornece uma série de ferramentas úteis para a análise de imagem, ou seja, é uma abordagem para análise de imagem, baseada na hipótese de que uma imagem consiste em estruturas que podem ser manuseadas através da teoria dos conjuntos. Baseia-se em ideias desenvolvidas por J. Serra e G. Matheron (SERRA; MATHERON, 1982). Para a morfologia matemática aplicada as imagens binárias, os conjuntos consistem das coordenadas (*x,y*) dos pixels (preto ou branco dependendo da convenção escolhida) da imagem. Outras apresentações úteis podem ser encontradas em (SERRA, 1986; HARALICK et al., 1987; FACON, 1996; GONZALEZ; WOODS, 2002). A Figura 2.1, descreve uma imagem binária e suas respectivas coordenadas.

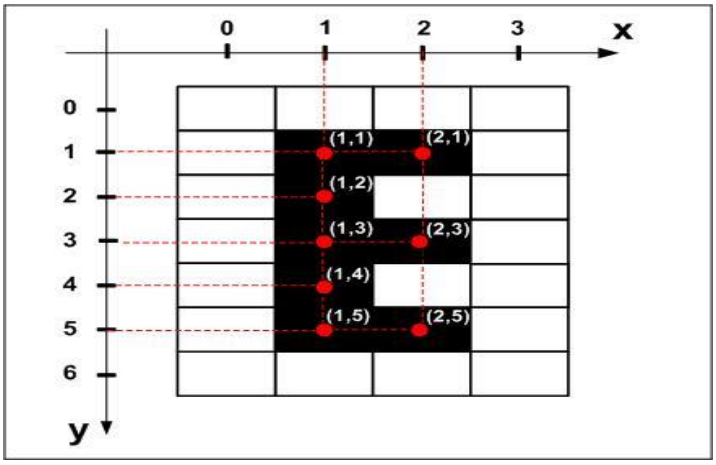



**Figura 2.1 – Descrição de uma imagem binária em formato de "E".**
**E = {(1,1); (1,2); (1,3); (1,4) ; (1,5); (2,1); (2,3); (2,5)}**

Com o auxílio da matemática, representada principalmente pela teoria dos conjuntos, é possível extrair de uma imagem inicialmente desconhecida às informações que dizem respeito a sua geometria com objetivos diversos, tais como: segmentação, realce, detecção de bordas, dentre outras (FACON, 1996). Basicamente, para que isto ocorra, é necessário que se utilize outro conjunto bem-definido denominado elemento estruturante (SE). Segundo Facon (FACON, 1996), um elemento estruturante é um conjunto com forma e tamanho conhecidos, que a partir de uma transformação pode ser comparado ao conjunto que represente uma imagem inicialmente desconhecida. O elemento estruturante é uma imagem binária pequena, ou seja, uma pequena matriz de *pixels*, cada um com um valor de zero ou um, onde: as dimensões da matriz especifica o tamanho do elemento estruturante; o padrão de uns e zeros especifica a forma do elemento estruturante; a origem do elemento estruturante é geralmente um dos seus *pixels*, embora geralmente a origem possa estar fora do elemento estruturante. Por isso, a escolha dele está diretamente relacionada com a forma geométrica do objeto a ser extraído numa imagem, entretanto, se tomarmos como base os operadores básicos como dilatação e erosão, pode-se dizer que o tamanho do elemento, define o quanto serão modificadas as características e os detalhes da imagem, ou seja, quanto maior o elemento estruturante, maior a área "erodida" ou "dilatada". Na Figura 2.2, são ilustrados alguns dos principais tipos de elementos estruturantes (SE) utilizados em processamento de imagens.

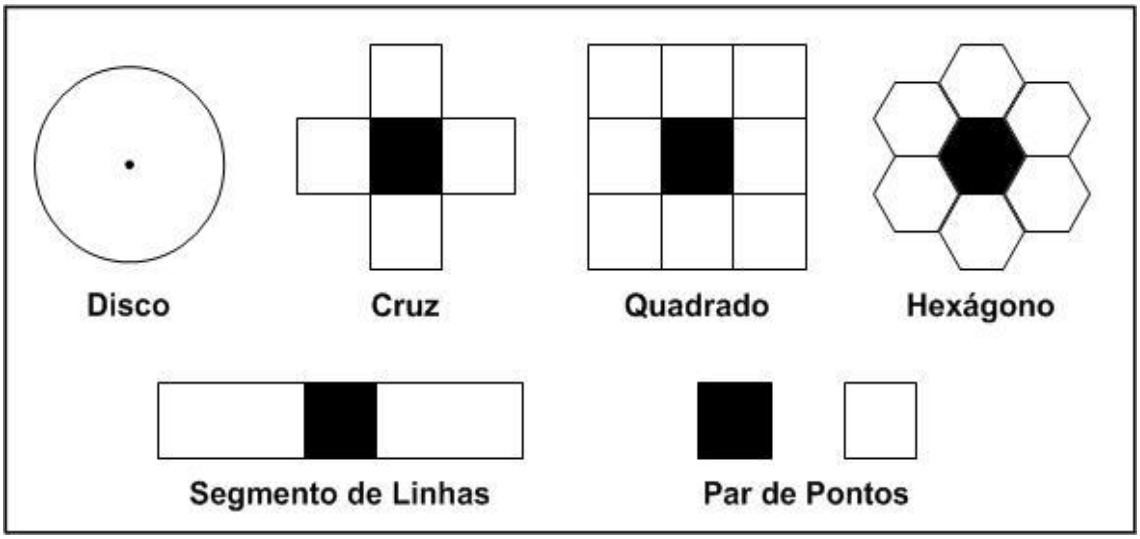



**Figura 2.2 – Tipos básicos de elementos estruturantes.**
**Fonte: Adaptado de (Soille, 1999).**

Segundo Matheron (MATHERON, 1975), extrair informações de uma imagem tem uma relação direta com a forma com que se observa. As Figuras que estiverem identificadas como “Fonte (MATLAB)” possuem imagens geradas utilizando a *toolbox* de processamento de imagem deste software.

### 2.1.1 Operadores Básicos da Morfologia Matemática

A Morfologia Matemática se caracteriza por um conjunto específico de operações sobre imagens, enquanto conjuntos de *pixels*. Dessas operações duas são consideradas como primárias (Dilatação e Erosão), pois essas são a base para a derivação de todas as outras operações morfológicas. Como citado anteriormente neste trabalho são abordadas as operações utilizadas em imagens binárias. A construção de sistemas morfológicos, geralmente, é implementada a partir da concepção do problema e pela escolha dos operadores mais adequados à solução em questão. Um dos maiores problemas para a adequação de operadores é observado na especificação dos elementos estruturantes. Uma possível solução para este problema é a criação de um mecanismo capaz de encontrar os elementos estruturantes “adequados”, de forma a realizar a transformação desejada.

### 2.1.1.1 Operador Morfológico de Dilatação

A dilatação de *A* por *B,* denotada $A \oplus B$, é definida como (GONZALEZ; WOODS, 2002):

$$\boldsymbol{A \oplus B = \{z|\left(\widehat{B}\right)_z \cap A \neq \emptyset\}} \quad \textbf{(2.1)}$$

Onde $\hat{B}$ representa a reflexão de *B* e $(\,)_z$ a translação de *z*. Assim, o processo de dilatação tem início com a obtenção da reflexão de $\widehat{B}$, em torno de sua origem, seguido da translação dessa reflexão por *z*. A dilatação de *A* por *B* é então, o conjunto de todos os deslocamentos *z* tais que a reflexão de *B* e *A* se sobreponham em pelo menos um elemento não nulo (PEDRINO, 2008). A Figura 2.3 ilustra uma aplicação do operador morfológico de dilatação.

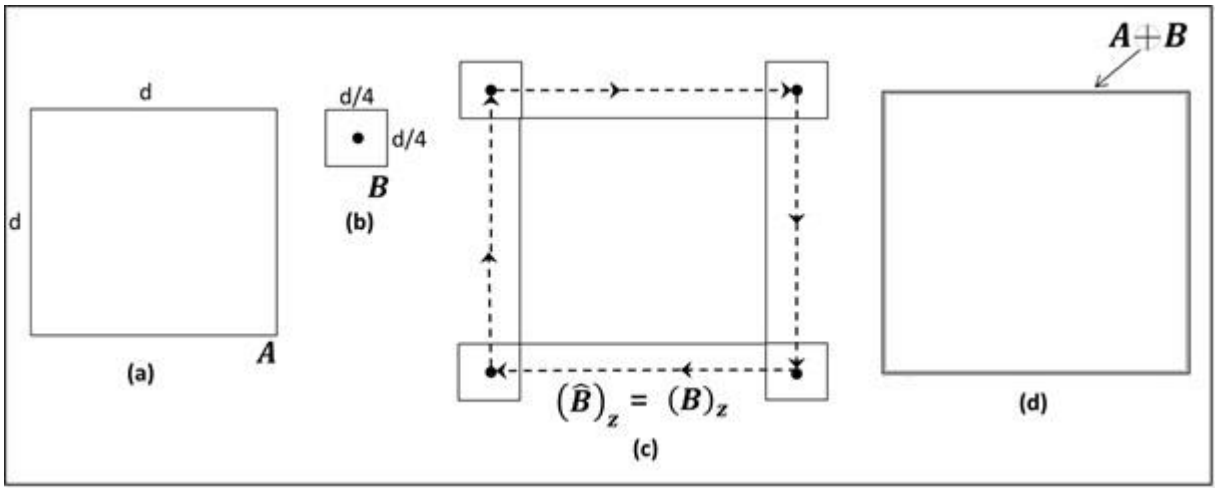



**Figura 2.3 – Demonstração do operador morfológico de dilatação.**
**Fonte: Adaptado de (GONZALEZ; WOODS, 2002).**

Todo o processo de dilatação apresentado na Figura 2.3, é formado por: (a) conjunto *A* (imagem original); (b) elemento estruturante do tipo quadrado (ponto de origem no centro); (c) conceito geométrico de deslocamento; (d) efeito causado pela dilatação "área sombreada" (imagem dilatada). Portanto, é possível perceber que o processo de dilatação tem a capacidade de ampliar os objetos da imagem, preencher lacunas menores e consequentemente unir objetos que estejam dentro de seu raio de atuação.

### 2.1.1.2 Operador Morfológico de Erosão

A erosão de *A* por *B*, denotada $A \ominus B$, é definida como (GONZALEZ; WOODS, 2002):

$$\boldsymbol{A \ominus B = \{z|(B)_z \subseteq A\}} \quad \textbf{(2.2)}$$

Deste modo, a erosão de *A* por *B* é o conjunto de todas as coordenadas dos pontos *z*, tais que *B*, quando transladado de *z*, fique contido em *A* (PEDRINO, 2008). Entretanto, o elemento estruturante *B* deve percorrer a imagem *A,* fazendo uma comparação entre os *pixels* que forem vizinhos de *z*. Caso o pixel de *B* seja correspondente à mesma posição *A* na vizinhança de *z*, este *pixel* é preservado. Na Figura 2.4 são ilustrados os passos da aplicação do operador morfológico de erosão.

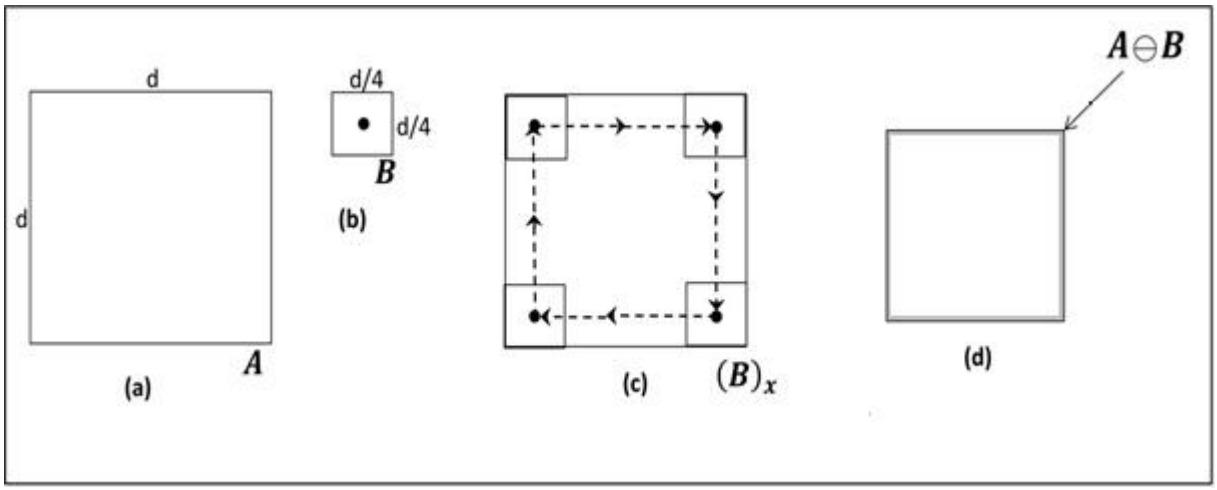



**Figura 2.4 – Demonstração do operador morfológico de erosão.**
**Fonte: Adaptado de (GONZALEZ; WOODS, 2002).**

O processo de erosão demonstrado na Figura 2.3 é formado por: (a) conjunto A (imagem original); (b) elemento estruturante do tipo quadrado (ponto de origem no centro); (c) conceito geométrico de deslocamento; (d) efeito causado pela erosão "área sombreada" (imagem erodida). As imagens da Figura 2.4 permitem a visualização da característica básica do operador de erosão, que se resume em diminuir objetos, eliminar objetos menores que o elemento estruturante utilizado, aumentar lacunas, permitindo assim a separação de objetos próximos. Enfim, é

importante ressaltar que todas as vezes que uma imagem é submetida a transformações de dilatação ou erosão, os conjuntos resultantes são sempre maiores ou menores que os originais respectivamente, ou seja, a dilatação aumenta e a erosão reduz. Por isso, os conceitos dos operadores de abertura e de fechamento que são abordados nas seções seguintes permitem ao usuário submeter uma imagem a uma filtragem sem que a mesma perca as suas principais características, no que diz respeito a sua forma e tamanho (conjuntos relevantes).

### 2.1.1.3 Operador Morfológico de Abertura

A abertura de um conjunto *A* por um elemento estruturante *B,* denotada $A \circ B$, é definada como (GONZALEZ; WOODS, 2002):

$$\boldsymbol{A \circ B = (A \ominus B) \oplus B} \qquad \textbf{(2.3)}$$

A operação de abertura geralmente funde intervalos estreitos e longos, elimina pequenos orifícios e preenche lacunas no contorno (GONZALEZ; WOODS, 2002). A Figura 2.5 ilustra a aplicação do operador morfológico de abertura.

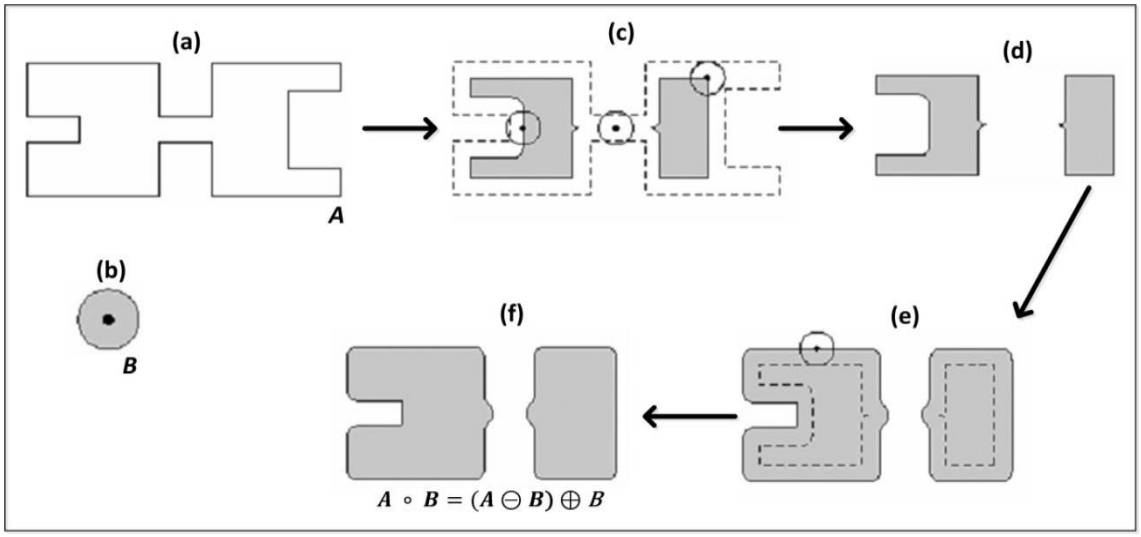



**Figura 2.5 – Demonstração do operador morfológico de abertura.**
**Fonte: Adaptado de (GONZALEZ; WOODS, 2002).**

Pode-se dizer então que o processo de abertura ilustrado na Figura 2.5 é representado por: (a) conjunto A (imagem original); (b) elemento estruturante do tipo “disco”; (c) processo do operador de erosão; (d) Imagem erodida; (e) processo de dilatação; (f) ação do operador de abertura “imagem aberta”. Assim, a operação de abertura é a utilização de uma transformação de erosão, seguida imediatamente por uma de dilatação, onde o resultado é a criação de um novo conjunto. Comparando a imagem inicial ilustrada na Figura 2.5, com a mesma após ter sofrido a ação deste operador, é possível perceber que os objetos menores em relação ao elemento estruturante foram totalmente eliminados, os contornos ficaram mais regulares e menos ricos em detalhes. Por isso, antes de utilizar este operador, é necessário levar em consideração que além das transformações citadas anteriormente, a imagem final não vai ter o mesmo conjunto inicial, pois os seus contornos vão estar nivelados pelo interior e algumas de suas partículas estarão separadas. Na seção seguinte um operador (fechamento) que produz um efeito contrário ao de abertura será descrito.

#### 2.1.1.4 Operador Morfológico de Fechamento

Para recuperar as estruturas modificadas pela dilatação em uma imagem, é utilizado o operador morfológico denominado de fechamento. Para uma imagem ser fechada pelo elemento estruturante é necessário efetuar na mesma primeiramente a transformação de dilatação seguida da erosão do resultado pelo mesmo elemento estruturante. Portanto, o fechamento de um conjunto *A* por um elemento estruturante *B,* denotado *A*●*B*, é definido como (GONZALEZ; WOODS, 2002):

$$\boldsymbol{A \bullet B = (A \oplus B) \ominus B} \qquad \textbf{(2.4)}$$

Este operador é dual ao operador de abertura, por isso, ele também tende a suavizar os contornos, funde as quebras em golfos finos, elimina pequenos buracos e preenche fendas em seus contornos (GONZALEZ; WOODS, 2002). A Figura 2.6 ilustra em detalhes a aplicação do operador morfológico de fechamento.

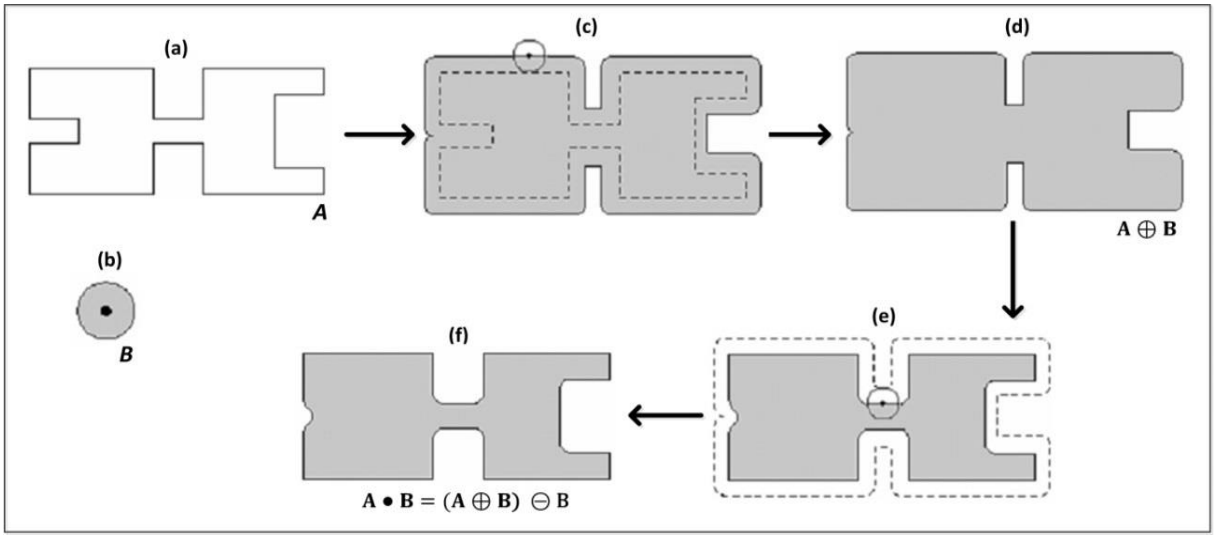



**Figura 2.6 – Demonstração do operador morfológico de fechamento.**
**Fonte: Adaptado de (GONZALEZ; WOODS, 2002).**

Assim, a operação morfológica de fechamento ilustrada na Figura 2.5 possui as seguintes imagens: (a) conjunto *A* (imagem original); (b) elemento estruturante do tipo “disco”; (c) processo do operador de dilatação; (d) Imagem dilatada; (e) processo de erosão; (f) ação do operador de fechamento “imagem aberta”. Ao contrário da abertura, a transformação de fechamento caracterizada na Figura 2.6, e que utilizou o mesmo tipo de elemento estruturante, mostra que as fronteiras foram suavizadas pelo exterior, os seus buracos foram preenchidos no interior das partículas que se mostraram menores que o elemento, as entidades que foram preservadas ficaram em sua maioria idênticas, o conjunto como um todo ficou mais regular e menos rico em detalhes.

### *2.1.1.5* Transformada *Hit-or-Miss*

A transformada *hit-or-miss* dentro da morfologia é considerada como uma ferramenta para reconhecimento de padrões, ou seja, tem o objetivo de encontrar a localização de uma determinada forma presente em uma imagem. Esta operação é caracterizada por utilização de dois elementos estruturantes $B_1$ e $B_2$ (a forma a ser reconhecida e seu respectivo “fundo”). As equações 2.5 e 2.6 representam esta operação (GONZALEZ; WOODS, 2002):

$$\boldsymbol{A \circledast B = (A \ominus B) \cap [\, A^c \ominus (W - X)\,]} \quad \textbf{(2.5)}$$

O objetivo desta operação é encontrar a localização (ponto de origem) de uma forma, que neste caso é o *X*. A erosão *(A ⊖ X)* pode ser vista geometricamente, como o conjunto de todas as localizações da origem de X, onde X encontre um acerto (hit) em *A* (GONZALEZ; WOODS, 2002). Na equação 2.5 *X* corresponde ao elemento estruturante $B_1$ e *(W – X)*, é equivalente a $B_2$.

$$\boldsymbol{A \circledast B = (A \ominus B_1) - (A \oplus \widehat{B}_2)} \quad \textbf{(2.6)}$$

Na Figura 2.7 são demonstrados os passos necessários para a execução da transformada “Hit-or-Miss”, denotada por *A⊛B.*

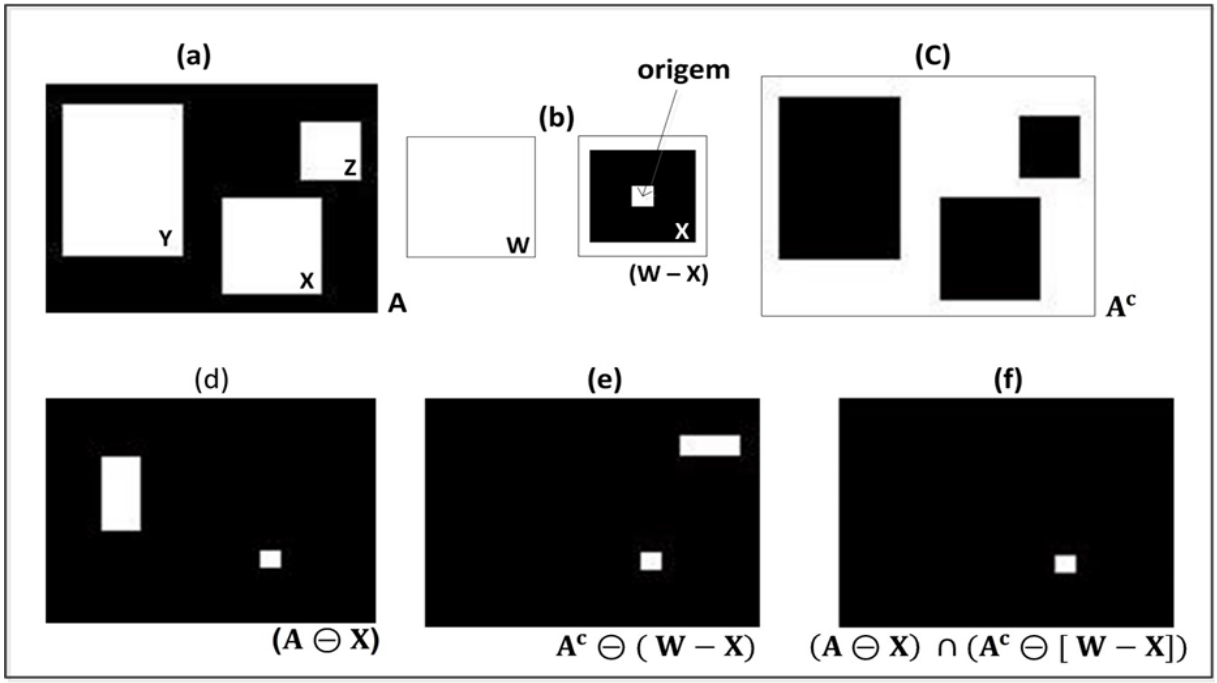



**Figura 2.7 – Demonstração do operador morfológico Hit-or-Miss.**
**Fonte: (MATLAB).**

A ilustração da Figura 2.7 está organizada da seguinte maneira: (a) conjunto *A* (imagem original); (b) a janela *W* e o fundo local de *X* que corresponde a (W – X); (c) o complemento do conjunto *A*; (d) a erosão de *A* por *X,* se os *pixels* das imagens do conjunto *A* corresponder exatamente aos *pixels* do elemento estruturante $B_1$, e se este estiver sob o elemento estruturante é definido com o valor zero, caso contrário, o

*pixel* será definido com o valor um, ou seja, mantem as imagens maiores ou iguais ao elemento estruturante; (e) a erosão de $A^c$ por *(W – X),* que ocorre o efeito contrário a de (d), onde as imagens menores e iguais são mantidas e a maiores eliminadas; (f) interseção de (d) e (e), representando a localização da origem de *X* desejada. A razão para a utilização de um elemento estruturante $B_1$ associado ao objeto e um elemento $B_2$ associado ao seu respectivo fundo, é baseada em uma definição baseada no princípio de que dois ou mais objetos são distintos, apenas se formarem conjuntos disjuntos (GONZALEZ; WOODS, 2002).

## 2.2 Conceitos Básicos da Computação Evolucionária

Embora a ideia básica de ver a evolução como um processo computacional foi concebida de várias formas durante a primeira metade do século XX, essas ideias realmente criaram raízes e começaram a se desenvolver na década de 1960. O catalisador foi o aumento na utilização de computadores digitais de baixo custo (pela comunidade científica), como uma poderosa ferramenta para modelagem e simulação. Vários grupos passaram a se interessar pela ideia de que modelos evolutivos, mesmo simples, poderiam ser expressos em formas computacionais e usados para resolver problemas mais complexos. Três ideias em particular, desenvolvidas também durante a década de 1960, serviram para definir e dar forma a esse campo então emergente: (a) utilizar processos evolutivos para resolver problemas de otimização com valor real de dificuldade (RECHENBERG, 1965); (b) alcançar os objetivos da inteligência artificial através de técnicas evolutivas (FOGEL et al., 1966); (c) processos evolutivos considerados como elementos-chave na concepção e implementação de sistemas adaptativos robustos capazes de lidar com um ambiente incerto e mutável (HOLLAND, 1962; HOLLAND, 1967). A especificação inicial e análise desses algoritmos evolucionários (EA’s) deixaram duas grandes questões não resolvidas: 1) como caracterizar o comportamento de sistemas passíveis de implementação; 2) compreender melhor como esses sistemas podem ser usados para resolver problemas computacionais. Como consequência, grande parte das pesquisas com algoritmos evolucionários na década de 1970 foram na

tentativa de obter dados adicionais sobre essas questões através de estudos empíricos e ampliações de teorias existentes. A maior parte dessa atividade foi desenvolvida pelos mesmos pesquisadores mencionados anteriormente, resultando no surgimento de três espécies distintas de “EA’s”: programação evolucionária (EP), estratégias evolucionárias (ES) e algoritmos genéticos (GA). Para melhor entendimento a respeito dos algoritmos evolucionários, na Tabela 2.1 são descritas as definições dos conceitos e na Figura 2.8 é ilustrado um diagrama básico do mesmo.

**Tabela 2.1 – Principais conceitos dos AG’s.**

| Conceitos | Definição |
|---|---|
| Espaço de Busca | Também conhecido como região viável, é o conjunto, espaço ou região que compreende as soluções possíveis ou viáveis para um determinado problema ser otimizado. |
| Cromossomo | Uma estrutura de dados responsável por codificar uma determinada solução para um problema, ou seja, um cromossomo tem a capacidade de representar um ponto dentro do limite do espaço de busca. |
| Gene | É considerado um parâmetro codificado no cromossomo, ou seja, um elemento do vetor que representa uma parte do cromossomo. |
| Alelo | Representa os valores que o gene pode assumir. Por exemplo, um gene que representa o parâmetro cor de um objeto poderia ter o alelo azul, preto, verde, vermelho etc. |
| População | Conjunto de cromossomos ou soluções na geração corrente. |
| Fenótipo | Representa o objeto, estrutura ou organismo construído a partir das informações do genótipo, ou seja, é o cromossomo decodificado. |
| Operadores Genéticos | Transformam a população através de sucessivas gerações, estendendo a busca até chegar a um resultado satisfatório. Sendo estes necessários para que a população se diversifique e mantenha características de adaptação adquiridas durante as etapas dos processos anteriores. |
| Descendentes | Nova prole, ou indivíduos gerados pelos “pais”. |
| Aptidão | Valor que quantifica a qualidade relativa (nível de adaptação) de um indivíduo, dada a função de avaliação. |
| Geração | Número de iterações que o algoritmo executa, ou seja a cada iteração é gerada uma nova geração. |
| Indivíduo | Um indivíduo é formado por um cromossomo dado e sua aptidão. |

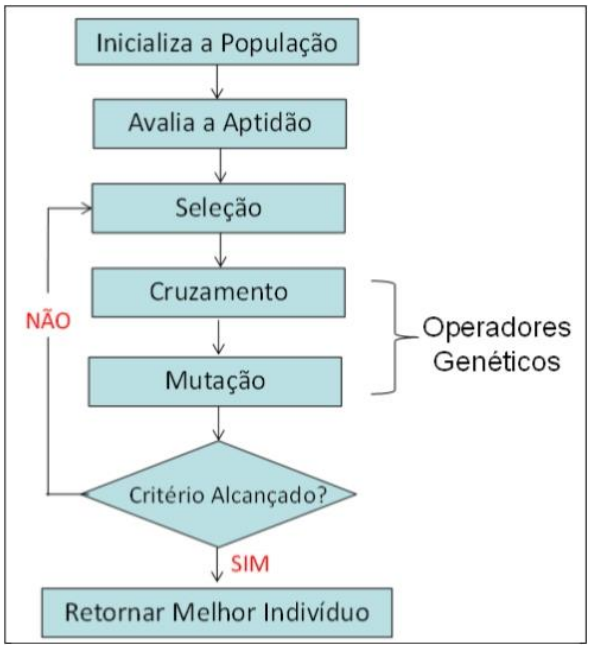



**Figura 2.8 – Diagrama Básico de uma Algoritmo Evolucionário.**

### 2.2.1 Programação Evolucionária

A programação evolucionária (EP) foi proposta pela primeira vez como uma abordagem para a inteligência artificial, sendo amplamente aplicada com sucesso em muitos problemas numéricos e de otimização combinatória (FOGEL et al., 1966). Mais tarde, no meio da década de 1980, David Fogel a usou para resolver tarefas mais gerais, incluindo problemas de previsão, otimização e aprendizagem de máquina (FOGEL, 1995).

As representações utilizadas na programação evolucionária são normalmente adaptadas para o domínio do problema (SPEARS et al., 1993). Uma representação comumente usada é um vetor de valores reais de comprimento fixo. A principal diferença entre tal programação e as outras abordagens evolucionárias (GA, GP e ES) é que não se utiliza o operador de recombinação entre os indivíduos da população, ou seja, utiliza-se apenas o operador de mutação.

Assim, a otimização por programação evolucionária clássica pode ser resumida em duas etapas principais: transformar as soluções candidatas (cromossomos) em uma população; e selecionar dentro dessa população os indivíduos que deverão ser mutados. Estes dois passos podem ser considerados como uma versão de base populacional, ou seja, o método de “geração e teste” clássico, no qual a mutação é

utilizada para gerar novas soluções (descendentes) e a seleção para testar quais dessas soluções (recém-criadas) devem sobreviver para fazer parte das gerações seguintes. Este campo de estudo ainda têm muitas publicações em diferentes áreas, como, por exemplo: processamento de imagens (LIU et al., 2012); e redes de computadores (redes sem fio) (SÁNCHEZ-MONTERO et al., 2012).

### 2.2.2 Estratégia Evolucionária

A estratégia evolucionária (ES) foi desenvolvida por Rechenberg e Schwefel através de alguns estudos nas décadas de 1960 a 1980 (RECHENBERG, 1965; RECHENBERG, 1973; SCHWEFEL, 1981). Uma estratégia evolutiva é um processo de reprodução contínua, avaliação e seleção. Cada nova geração é uma melhoria sobre o que aconteceu antes. Isto resulta em sistemas que se tornam cada vez mais eficientes e mais organizados.

Enquanto a programação evolutiva derivou para o interesse puramente científico, a motivação desse estudo foi, desde o início, resolver problemas de projetos de engenharia. Suas características mais importantes são: (1) codificação real dos parâmetros, uma vez que o modelo de evolução orgânica acontece a nível individual; (2) depende da seleção determinística e da mutação para sua evolução; (3) parâmetros estratégicos de mutabilidade para adaptação automática.

Em estratégias evolucionárias, uma representação é utilizada como um vector de valores reais e de comprimento fixo, onde cada posição do vetor corresponde a uma característica do indivíduo. No entanto, essas características são consideradas como comportamentais e não estruturais. Recentemente, tal técnica ainda é muito utilizada em trabalhos de: otimização de funções lineares e não lineares (LI; QING-LAN, 2012), utilização de operadores de cruzamento híbridos (XIAODONG et al., 2011), só para citar alguns.

### 2.2.3 Algoritmos Genéticos

Os algoritmos genéticos (GA's) devem seu nome a uma ênfase inicial em representar e manipular os indivíduos em nível de genótipo. No trabalho original de John Holland (HOLLAND, 1975), os GA's foram propostos para compreender os

fenômenos de adaptação em ambos os sistemas, naturais e artificiais, onde têm três características consideradas como chave que se distinguem de outros métodos computacionais inspirados na evolução natural: o uso de uma sequência de *bits* para a sua representação; a utilização da seleção proporcional; o uso do método primário de cruzamento (*crossover*) para a produção de variantes. Algoritmos Genéticos são as técnicas mais populares nas pesquisas de computação evolutiva, onde na representação é utilizada uma sequência de bits com comprimento fixo. Cada posição nessa sequência é a representação de uma característica particular de cada indivíduo e o valor armazenado na posição representa a forma como tal característica é expressa na solução. Normalmente, a sequência é avaliada como um conjunto de características estruturais de uma solução que tem pouca ou nenhuma interação. A analogia pode ser estabelecida diretamente com os genes de organismos biológicos. Cada gene representa uma entidade que é estruturalmente independente de outros genes. Os GA's são muito utilizados em vários campos de pesquisa, sendo que dois dos mais recentes são: uso para diagnósticos de osteoporose (JENNANE et al., 2010); e processamento de imagens (sensoriamento remoto) (DONGJIAN et al., 2010).

### 2.2.4 Programação Genética

Programação Genética (GP) é a técnica mais popular para programação automática e fornece um contexto adequado para geração automática de procedimentos de imagem (KOZA, 1992). É um ramo da computação evolucionária e Inteligência Artificial (AI), baseado em conceitos de genética e do princípio da seleção natural de Darwin, que tem como finalidade produzir e desenvolver programas de computador para resolver problemas diversos. Como tal, a GP pode ser considerada como uma extensão dos algoritmos genéticos. São programas (no sentido de sequência de instruções), que são considerados como soluções candidatas para um problema particular. A representação das populações é abordada como contendo elementos de um conjunto de funções (operadores aritméticos (+, -, * etc.), funções matemáticas (seno, log etc.), operadores lógica (*And, Or* etc.)) aliada ao conjunto de terminais (composto pelas variáveis, constantes e funções de aridade zero sem argumentos) que, devidamente combinados de forma sintática e semanticamente, representa a solução de problemas no domínio de interesse. Esses algoritmos têm

provado ao longo dos anos terem se tornado uma ferramenta poderosa como uma estratégia de busca em diversos problemas, como por exemplo, pode-se citar: uma abordagem para agrupar de forma otimizada os sinais dos componentes de usinas nucleares (combustível nuclear, fluido refrigerante, barras de controle e segurança, refletor e moderador) (BARALDI et al., 2011) e processamento de imagens (PEDRINO et al., 2011). Essa subárea da computação evolucionária serviu de embasamento teórico para dar origem à outra subárea conhecida como programação genética cartesiana, que é a técnica utilizada em todo o desenvolvimento deste projeto de mestrado.

### 2.2.5 Programação Genética Cartesiana

A programação genética cartesiana (CGP) foi originalmente desenvolvida por J. Miller a partir de um método para evolução de circuitos digitais (MILLER et al.,1997). No entanto, o termo “programação genética cartesiana” apareceu pela primeira vez em um trabalho no ano de 1999 (MILLER, 1999) e foi proposta como uma forma geral de programação genética em 2000 (MILLER; THOMSON, 2000). Nos últimos anos, a forma clássica da CGP, foi aprimorada de várias formas, incluindo as funções definidas automaticamente. Mais recentemente, foi desenvolvido por Harding, Miller e Wolfgang um trabalho denominado auto modificação, que inclui funções primitivas que modificam o programa em nível de fenótipo, aumentando-se assim sua eficiência (HARDING et al., 2010). A CGP pode ser utilizada em diversas áreas como: aprendizagem de máquina; redes neurais; inteligência artificial; mineração de dados; previsão financeira, otimização de funções, classificação, desenho de circuitos eletrônicos, diagnósticos médicos, dentre outras.

#### 2.2.5.1 Forma Geral da CGP

A denominação “cartesiana” é devido ao fato do programa ser representado por uma grade bidimensional (*grid*) de nós, ou seja, grafos acíclicos direcionados (*DAG - Directed Acyclic Graph*). Um dos benefícios deste tipo de representação é a reutilização implícita dos nós pertencentes ao grafo direcionado.

O genótipo em CGP tem um comprimento fixo. No entanto, o tamanho do fenótipo (em termos de números de nós computacionais) pode ser qualquer coisa

desde zero nós, até o número de nós definidos anteriormente no genótipo. Os tipos de funções dos nós computacionais utilizados em CGP, são decididas pelo usuário e são listadas em uma tabela *look-up table* (LUT). Em CGP, cada nó no grafo orientado representa uma função específica que é codificada por certo número de conexões de entrada dos genes.

Esta técnica utiliza um mapeamento do tipo genótipo-fenótipo, e não requer necessariamente que todos os nós devam estar ligados uns aos outros, resultando assim, em um fenótipo com sua variação de tamanho limitada. Isto permite que algumas áreas (nós) do genótipo fiquem inativas, ou seja, não tenham nenhuma influência sobre o fenótipo, levando-se assim, a um efeito de neutralidade que é também conhecido como redundância genética (veja subseção 2.2.5.2.4). Um gene codifica uma função que representa um nó, e o restante dos genes codificam de onde o nó obtém suas entradas no grafo.

Os nós devem receber as suas entradas por uma alimentação dirigida para frente a partir de qualquer um dos nós pertencentes a uma coluna anterior, ou a partir de uma das entradas do programa ($n_i$), que é geralmente chamada de terminal, ou seja, os nós da mesma coluna não podem ser conectados uns aos outros. Além disso, o número de entradas, que tem um nó, é determinado pela *aridade* da função (número de argumentos ou operandos tomados) que representa o mesmo.

Nas entradas de dados do programa são apresentados dados com endereços absolutos *0* à (*n-1*), onde *n* é o número de entradas do programa. As saídas de dados dos nós de um genótipo são representadas por dados com endereços sequenciais, coluna por coluna, iniciando a partir de *n* até (*n* + (*m-1*)), onde *m* é utilizado para determinar o limite superior do número de nós (igual ao número de linhas multiplicado pelo número de colunas).

Se o problema exige *k* saídas do programa ($n_o$), então são adicionados *k* inteiros ao fim do genótipo, cada um sendo um endereço absoluto da saída de um dos nós da saída, onde é determinado o resultado do programa. A forma geral da programação genética cartesiana é ilustrada na Figura 2.8.

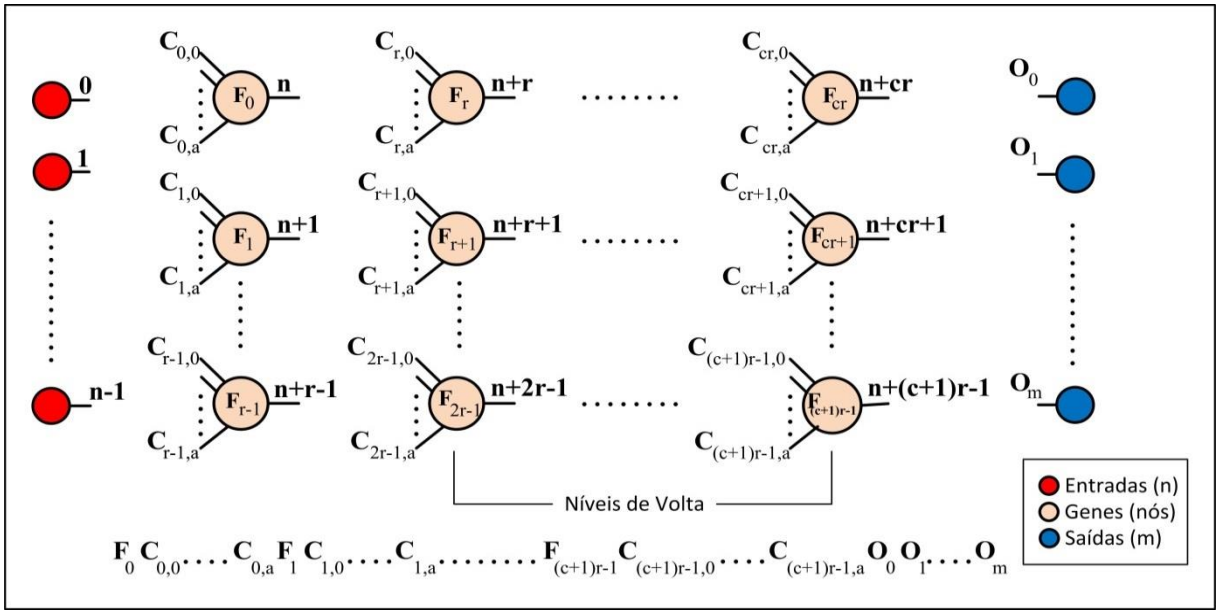



**Figura 2.9 – Forma geral da CGP.**

**Fonte: Adaptado de (MILLER, 2011).**

A forma geral bidimensional da CGP trata-se de uma grade de nós cujas funções são escolhidas a partir de um conjunto de funções primitivas. Dois parâmetros *c*, e *r*, respectivamente, definem o número de colunas e linhas da grade bidimensional. Cada nó assume ter um número máximo de entradas baseado em sua função de aridade *a.* Todos os dados de entrada e saída dos nós são identificados consecutivamente (iniciando em zero), o que lhe garante um endereço único, que especifica de onde os dados de entrada ou o valor de saída do nó podem ser acessados (entradas(n), saídas(m)). Um nó pode ter apenas as suas entradas ligadas a ambos os dados de entrada ou a saída de um nó em uma coluna anterior. Em geral, pode haver certo número de genes de saída (*Oi*), que especifique de onde as saídas do programa são retiradas. Todos os nós (genes) de uma função $F_i$ são endereços representados por números inteiros em uma *look-up table* de funções. Todos os genes de conexão $C_{ij}$, são endereços de dados absolutos tendo valores inteiros de *0* até o endereço do nó na parte inferior da coluna anterior. A CGP possui três parâmetros que são escolhidos pelo usuário: número de colunas ($n_c$); número de linhas ($n_r$) e níveis de volta (*l*). A partir do número de linhas e de colunas é possível determinar o número máximo de nós permitidos $L_n = n_c * n_r$. O parâmetro *l determina* a conectividade entre os nós, ou seja, restringe as colunas que um nó pode obter os seus dados de entrada, por exemplo: *l = 1* um nó só pode obter suas entradas a partir de um nó

pertencente à coluna imediatamente a sua esquerda, ou da entrada primária; *l = 2* um nó pode receber suas entradas a partir dos nós das duas colunas a sua esquerda ou da entrada primária; $l = n_c$ um nó pode receber suas entradas de todas as colunas a sua esquerda e da entrada primária. Portanto, a variação desses parâmetros pode resultar em vários tipos de topologias para o grafo, veja Figura 2.9.

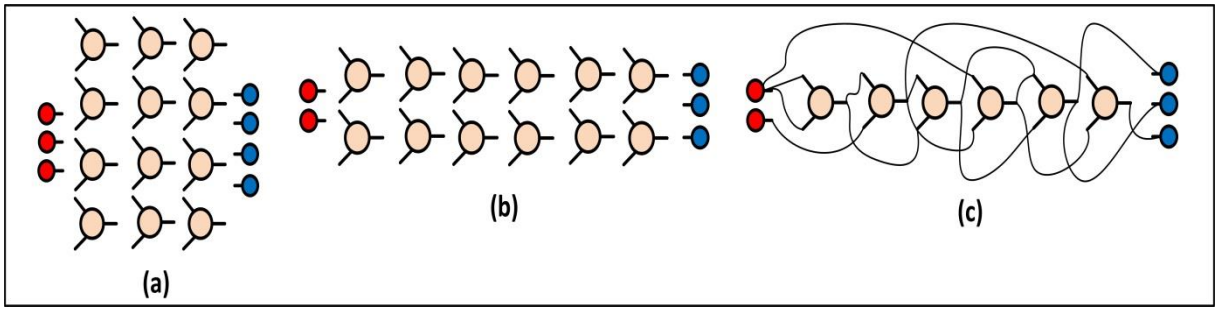



**Figura 2.10 – Algumas topologias em CGP.**

**Fonte: Adaptado de (MILLER, 2013).**

### 2.2.5.1.1 Exemplos

Em CGP podem-se representar muitos tipos de estruturas computacionais como: circuitos digitais; equações matemáticas; geração de imagens, sempre baseado na variação dos parâmetros (linhas, colunas, conectividade) e das funções (aritméticas, booleanas, morfológicas, entre outras). Na Figura 2.10 é demonstrado um exemplo de genótipo codificado e seus respectivos parâmetros e funções.

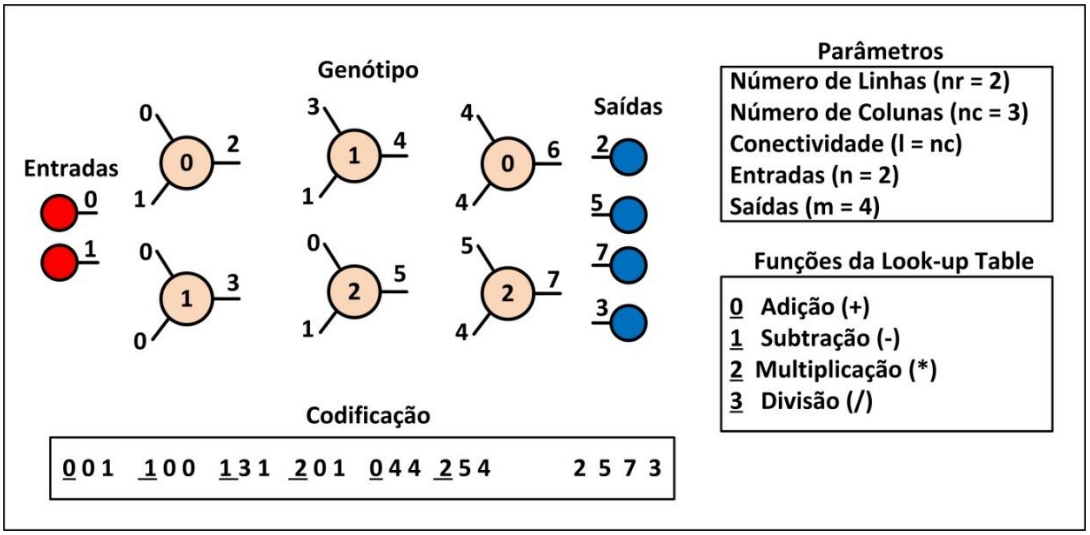



**Figura 2.11 – Exemplo de um genótipo codificado em CGP.**

**Fonte: Adaptado de (MILLER, 2013).**

Este exemplo mostra um genótipo codificado em CGP. Este vai ser representado utilizando os seguintes parâmetros: número de colunas ($n_c = 3$); número de linhas ($n_r = 2$); conectividade ($l = n_c$). É possível perceber que estão sendo utilizadas duas entradas ($n$) e quatro saídas ($m$). A *look-up table* é composta por quatro funções primitivas (adição, subtração, multiplicação e divisão). A codificação deste genótipo é representada por uma lista de inteiros, sendo que todos os nós são compostos por três dígitos, onde o primeiro número sublinhado representa a função (operação), o segundo representa a entrada superior e o terceiro a entrada inferior de cada nó. A saída de cada nó tem a sua numeração sequencial iniciando com (n+1), e o número de saídas é igual a quatro (m=4). É importante ressaltar que a conectividade neste caso é total, ou seja, todos os nós podem receber dados de todas as colunas anteriores (à esquerda) e das entradas $n$. A Figura 2.11 está dividida em três partes: (a) obtenção do grafo através das conexões que se iniciam a partir das duas entradas (terminal), obedecendo à ordem dos números que representam a saídas de cada nó; (b) codificação do grafo com uma lista de números inteiros; (c) representação do grafo, após as entradas dos nós serem submetidas à ação das funções primitivas.

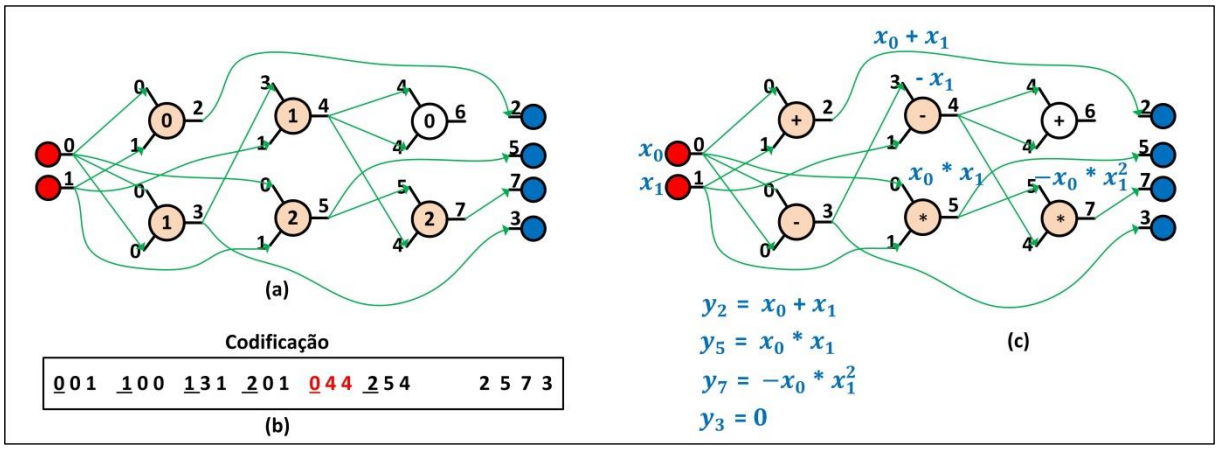



**Figura 2.12 – Obtenção do grafo e sua representação em CGP.**

**Fonte: Adaptado de (MILLER, 2013).**

O nó que possui sua saída identificada com o número 6, não está conectado a nenhuma entrada de nós, nem nas saídas do programa. Esses nós em CGP são conhecidos por estarem silenciosos, ou seja, não são codificados e com isso apresentam pouca sobrecarga computacional.

### 2.2.5.2 Mapeamento Genótipo - Fenótipo em CGP

Ao decodificar um genótipo, muitos nós e os seus dados podem ser ignorados, pois, eles não são referenciados no caminho de entradas e saídas. Estes dados podem ser alterados e não vão fazer diferença em nível de fenótipo.

#### 2.2.5.2.1 Restrições Alélicas

Os valores que os genes podem assumir (isto é, alelos) em programação genética cartesiana são extremamente restritos, por exemplo, é preciso diferenciar o que é uma entrada de dados, e o que é funções (operadores) pertencentes aos genes. Esta restrição deve ser obedecida, principalmente quando os genótipos são submetidos a operadores como recombinação (*crossover*) ou mutação. Para que esta restrição seja garantida, é necessário primeiramente que os alelos da função genes $f_i$, tenham valores e endereços válidos em relação à *look-up table* (onde estão armazenadas as funções primitivas). Como $n_f$ representa o número de funções genes permitidas, então $f_i$ deve obedecer a seguintes condições:

$$\mathbf{0} \leq \boldsymbol{f_i} \leq \boldsymbol{n_f} \tag{2.7}$$

Considerando-se um nó que esteja localizado na coluna *j*. Os valores assumidos pela conexão $C_{ij}$ do mesmo devem obedecer:

Se $j \geq l$:

$$\boldsymbol{n_i} + (\boldsymbol{j} - \boldsymbol{l})\, \boldsymbol{n_r} \leq \boldsymbol{c_{ij}} \leq \boldsymbol{n_i} + \boldsymbol{jn_r} \tag{2.8}$$

Se $j < l$:

$$\mathbf{0} \leq \boldsymbol{c_{ij}} \leq \boldsymbol{n_i} + \boldsymbol{jn_r} \tag{2.9}$$

Por fim as saídas $O_i$ podem conectar-se a qualquer nó ou entrada primitiva, veja:

$$\mathbf{0} \leq \boldsymbol{O_i} \leq \boldsymbol{n_i} + \boldsymbol{L_n} \tag{2.10}$$

### 2.2.5.2.2 Mutação

Dentre os possíveis tipos de mutação (*bit-flip*, fronteira, não uniforme, uniforme e o gaussiano) em CGP, a mais utilizada é a mutação determinada por um operador de ponto, ou seja, na mutação pontual um alelo é escolhido aleatoriamente e logo em seguida é alterado para um valor também válido e aleatório. Se uma função, representada por um gene é escolhida para mutação, a mesma pode receber um dado vindo de qualquer endereço pertencente ao domínio das funções. No caso deste gene ser de entrada, ele pode receber quaisquer dados vindos de outro nó, anterior ou das entradas do programa. O número de nós de um grafo que pode ser escolhido para mutação em uma única aplicação, deve ser definido pelo usuário, sendo que este deve obedecer a uma porcentagem do valor total de genes num cromossomo (definido de acordo com a necessidade do resultado). Este número é chamado de "taxa de mutação", denotado pelo $\mu_r$. Isto quer dizer que esse valor se refere ao número efetivo de nós que podem ser submetidos à mutação em um cromossomo com comprimento $L_g$. Para essa quantidade é utilizado o símbolo $\mu_g$, portanto $\mu_g = \mu_r L_g$. Na Figura 2.13 apresenta-se um exemplo de mutação pontual.

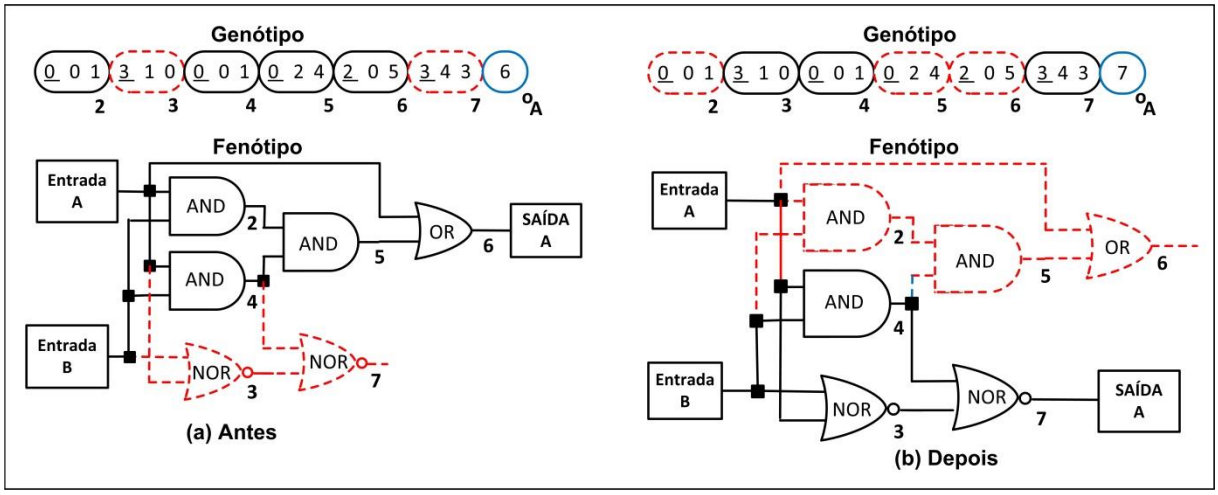



**Figura 2.13 – Ilustração de mutação pontual em CGP.**

**Fonte: Adaptado de (MILLER, 2011).**

O exemplo do operador de mutação pontual, ilustrado na Figura 2.13, destaca como uma pequena mudança no genótipo (mutação somente da saída) pode às vezes produzir uma grande mudança no fenótipo. É possível perceber que nas duas partes,

(a) e (b), as mesmas possuem pontos destacados na cor vermelha, o que significa genes inativos (redundância), e as saídas destacadas na cor azul, que é onde ocorre a mutação. As duas imagens pertencentes à Figura 2.13 representam: (a) um genótipo devidamente codificado e seu respectivo fenótipo antes da mutação; (b) o genótipo modificado pela ação do operador de mutação e o novo fenótipo gerado após essa mutação. Portanto, todo processo, inicia-se com a mutação do endereço de saída $o_A$, que é representada pelo número *6* (Figura 2.13(a)), para o valor *7* (Figura 2.13(b)), como consequência, os nós *3* e *7* (Figura 2.13(a)) que na configuração anterior estavam inativos, passam a ser ativos, diferentemente dos nós 2, 5 e 6, que precisam estar ativos para o fenótipo produzir o resultado na nova saída *7*. Assim, segundo Miller (MILLER, 2011), o operador de mutação em geral tem por finalidade efetuar uma troca no conteúdo de uma posição do genótipo, seja nos alelos ou nas funções primitivas, sempre com uma determinada probabilidade de no máximo 10%.

#### 2.2.5.2.3 Recombinação (*crossover*)

Mais conhecida como cruzamento, a recombinação tem a capacidade de produzir um par de filhos (veja, Figura 2.13) a partir de um par de indivíduos (pais), ou seja o filhos são partes idênticas retiradas dos pais. A escolha de cada indivíduo (pai) é mediante a probabilidade proporcional à sua aptidão normalizada sempre respeitando o método de seleção escolhido.

Esta operação é iniciada logo que um ponto de cada indivíduo é escolhido, utilizando uma distribuição uniforme de acordo com uma dada probabilidade. Este ponto corresponde ao gene que é utilizado como base para a escolha dos outros que serão utilizados na operação. Ambos os filhos são resultado de uma substituição de alelos (determinado por probabilidade) do primeiro indivíduo pela sequência de alelos escolhida do segundo pai. É importante ressaltar que esse processo de criação é utilizado para gerar os dois filhos.

A Figura 2.14, representa o processo de cruzamento aplicado a dois pais, onde o filho 1 recebeu quatro alelos do pai 1 e oito alelos do pai 2, conforme ilustrado na imagem denominada Genótipo 1(filho). Já o filho 2 recebeu uma sequência diferente (oito alelos do “pai 2 “e quatro alelos do “pai 1”).

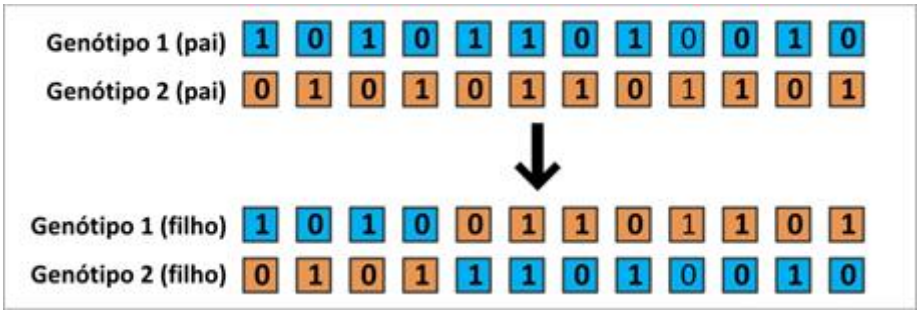



**Figura 2.14 – Exemplo de cruzamento em CGP.**

#### 2.2.5.2.4 Redundância (neutralidade)

Num genótipo CGP podem haver genes que são inativos (uma forma de redundância), e que não exercem nenhuma influência sobre o fenótipo e consequentemente têm um efeito neutro sobre a aptidão. Este fenômeno é identificado como uma neutralidade, onde a representação possui genes que podem ser ativados ou desativados por operadores de mutação durante a evolução. Dependendo da aplicação, os genótipos são formados em sua maioria por genes redundantes. Miller e Smith (MILLER; SMITH, 2006), buscando investigar dentre outras características e propriedades da CGP, o papel e a utilidade da redundância na busca evolutiva, produziram um trabalho, na qual, foi utilizado um multiplicador de dois *bits* (calculando o produto de dois números binários de dois *bits)*, com os seguintes parâmetros: quatro entradas digitais; quatro saídas binárias; conjunto de funções primitivas {AND, OR, NAND, NOR}. Os gráficos mostrados na Figura 2.14, representam: (a) comprimento médio do fenótipo ao final da busca evolucionária, para todas as probabilidades de mutação em relação ao comprimento do genótipo (calculado pelo número de nós), representando um multiplicador de dois bits com um conjunto de portas lógicas {AND, OR, NAND, NOR}; (b) proporção média de nós ativos em um genótipo após a conclusão da evolução para todas as probabilidades de mutação em relação ao comprimento do genótipo (calculado pelo número de nós) para um multiplicador de dois bits com o mesmo conjunto de portas lógicas {AND, OR, NAND, NOR}. O genótipo tem o comprimento de 4000 genes, na qual, 5% dos nós são ativos e 95% são inativos. Assim, segundo Miller e Smith (MILLER; SMITH, 2006), é possível determinar que o esforço computacional seja menor ao mesmo tempo em que a neutralidade é maior.

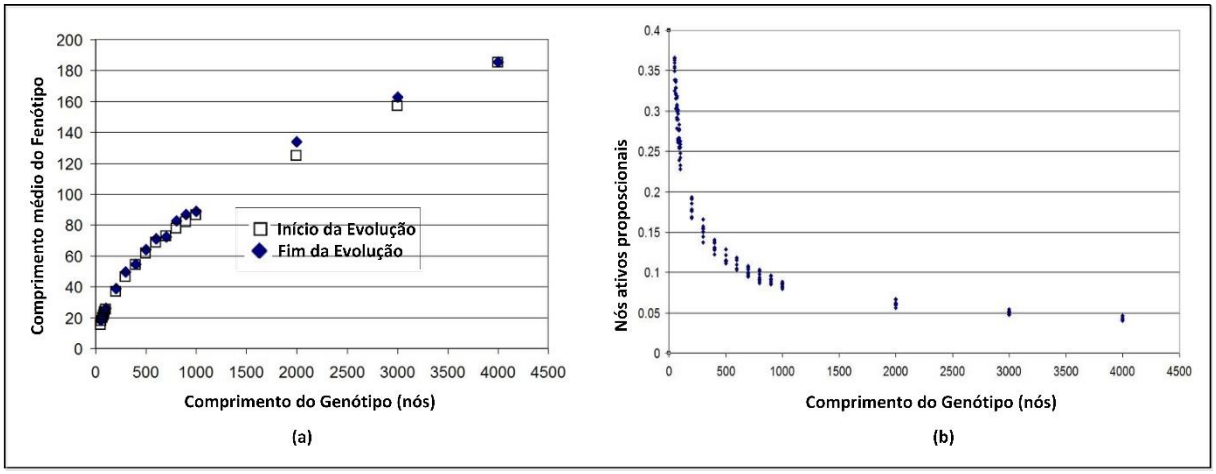



**Figura 2.15 – Gráficos de redundância em CGP.**
**Fonte: Adaptado de (MILLER; SMITH, 2006).**

## 2.3 Considerações Finais

Neste capítulo foram apresentados os conceitos básicos dos principais temas que envolvem este projeto de mestrado. Inicialmente foi abordada a teoria de morfologia matemática, que é muito utilizada em processamento de imagens e fornece alguns operadores utilizados neste projeto. A fundamentação teórica relacionada à computação evolucionária antecede a principal abordagem desta revisão, que é denominada de programação genética cartesiana (CGP). Esta técnica será à base para implementação do projeto de mestrado.

# Capítulo 3

# *REVISÃO RELACIONADA*

*Neste capítulo será realizada uma revisão relacionada à programação genética cartesiana utilizada para processamento de imagens, iniciando-se com a seção 3.1 de Considerações Iniciais, em seguida, descrevendo-se os trabalhos relacionados com este projeto de mestrado (seções, de 3.2 á 3.5) e também o artigo base que motivou a elaboração deste projeto (seção 3.6), e por fim, o artigo utilizado como base de comparação dos resultados encontrados nos experimentos (seção 3.7).*

## 3.1 Considerações Iniciais

Como citado anteriormente, este projeto de mestrado utiliza uma abordagem conhecida como programação genética cartesiana para construção de filtros automáticos para processamento de imagens binárias. Assim, para fundamentar esta proposta de trabalho, nesta revisão, são apresentadas algumas descrições sucintas de trabalhos relacionados (tanto em *software*, quanto em *hardware*) que também buscaram encontrar na programação genética cartesiana e em sua antecessora, a programação genética, uma maneira mais simples e eficiente de diminuir o esforço computacional por meio desta automatização. Portanto, nesta seção serão detalhados os seguintes trabalhos:

1. Programação Genética Cartesiana para Processamento de Imagens (*CGP-IP – Cartesian Genetic Programming for Image Processing),* que é uma abordagem para vários tipos de problemas em diferentes domínios (processamento básico de imagem, imageamento médico e detecção de objetos em robótica), onde, se utiliza uma mistura de operações matemáticas e primitivas de alto nível (HARDING et al., 2012);

2. Classificação do Terreno de Marte usando Programação Genética Cartesiana, que também utiliza a técnica de CGP-IP para classificação automática do terreno, mais precisamente as pedras, a areia e o cascalho (LEITNER et al., 2012);
3. Programação Genética Cartesiana para Tarefas de Processamento de Imagens (*Cartesian Genetic Programming for Image Processing Tasks*), onde são apresentados os resultados experimentais para análise de imagens, ou seja, a eficiência dessa abordagem é analisada para um problema envolvendo a localização de objetos numa imagem (MONTES; WYATT, 2003);
4. Projeto de um Circuito Polimórfico para Reconhecimento Rápido de Caracteres Manuscritos usando CGP (*Polymorphic Circuit Design for Speedy Handwritten Character Recognition Using Cartesian Genetic Programming*), para esta abordagem as imagens são convertidas para a forma binária e dispostas de uma dimensão de matriz, utilizando um circuito muito mais simples com multiplexadores de 1-bit (REHMA; KHAN, 2011);
5. Projeto e Implementação de uma Arquitetura Reconfigurável Virtual para Diferentes Aplicações de Hardware Evolutivo Intrínseco (*VRA – Design and Implementation of a Virtual Reconfigurable Architecture for Different Applications of Intrinsic Evolvable Hardware*), onde se implementa em (FPGA – *Field Programmable Gate Array*) e específica para *hardware* evolutivo intrínseco (*EHW – Intrinsic Evolvable Hardware*), na qual é dedicada ao reconhecimento de caracteres e operadores de imagem (WANG et al., 2008);
6. Sistema Baseado em GP para a Construção Automática de Filtros de Imagem (*IFbyGP – A Genetic Programming Based System for the Automatic Construction of Image Filters*), que uma proposta para a construção automática de uma sequência de operadores morfológicos, convolução e operadores lógicos, ao qual foi desenvolvida para processamento de imagens binárias, reconhecimento de notas musicais, processamento de imagens em nível de cinza e reconhecimento de laranjas em um pomar.

## 3.2 CGP para Processamento de Imagens (HARDING et al., 2012)

Combinando o conhecimento de domínio tanto sobre o processamento de imagens e de técnicas de aprendizagem de máquina, é possível expandir as capacidades da Programação Genética dentro do domínio de processamento de

imagens. Por isso, essa técnica denominada de CGP-IP permite a geração automática de programas de computador usando as funcionalidade especificas de processamento de imagens da biblioteca OpenCV (BRADSKI, 2000). Na Figura 3.1 é ilustrado um exemplo de um genótipo devidamente mapeado pelas regras do CGP-IP determinada para tal experimento.

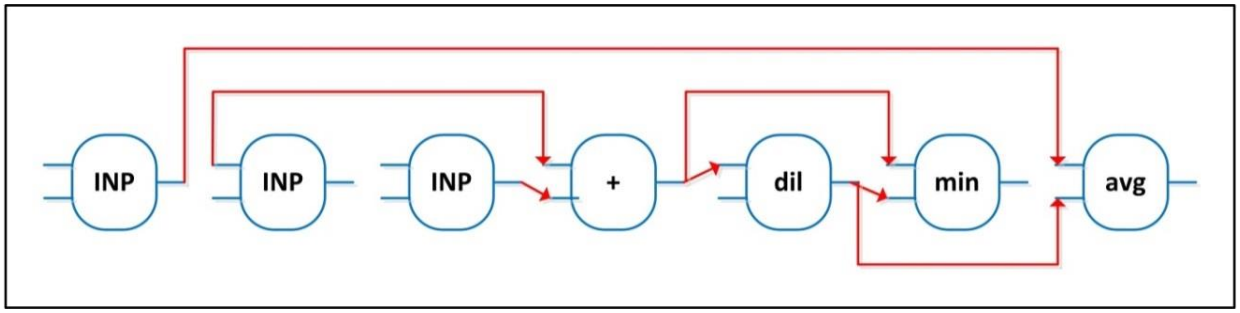



**Figura 3.1 – Ilustração de um genótipo em CGP-IP**

**Fonte: Adaptado de (HARDING et al., 2012).**

### 3.2.1 Experimento e Resultados

Apesar do trabalho sobre CGP-IP apresentar mais experimentos em alguns domínios específicos e ter usado em todos eles a mesma configuração (com exceção da função de aptidão, que deve ser apropriada para cada caso), nesta subseção é descrito apenas um dos experimentos com foco na área médica: detectar e contar células cancerosas submetidas à mitose, ou seja, se trata de um caso de reconhecimento de padrões. Para testar o problema com o CGP-IP, o conjunto de treinamento foi montado com pequenas partes da imagem original, ou seja, uma metade da imagem contém uma ou mais mitoses e a outra metade contém espaços vazios selecionados aleatoriamente. Segundo Harding (HARDING et al., 2012), no total, 420 imagens foram usadas para correções, com 356 utilizadas para treinamento e as 64 restantes para validação. Como se trata de produzir uma classificação binária, usando a função de aptidão baseada em *MCC (Matthews Correlation Coefficient*), que é o coeficiente de correlação de Matthews (MATTHEWS, 1975), denotada com a seguinte equação:

$$\mathbf{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad \textbf{(3.1)}$$

Os resultados são baseados na análise por *pixel. A*ssim, com a análise adicional de segmentos, a taxa de classificação de mitose também pode ser determinada. Segundo Harding (HARDING et al., 2012), de 42 mitoses o CGP-IP identificou corretamente 36 delas, que é equivalente a 86% do total. Deste número foram apresentados 12 falso-positivos e 6 falso-negativos. Os resultados estatísticos para os 6 testes executados com o CGP-IP são apresentados na Tabela 3.1.

**Tabela 3.1 – Resultados estatísticos da classificação e aptidão.**

| | **Precisão de Classificação (%)** | **MCC (Aptidão)** |
|---|---|---|
| Média | 98 | 0,36 |
| Mínimo | 97 | 0,28 |
| Máximo | 98 | 0,46 |
| Desvio-Padrão | 0.3 | 0,08 |

Na Figura 3.2 são ilustradas a imagem de entrada, a imagem prevista e a imagem classificada de acordo com a validação do melhor indivíduo.

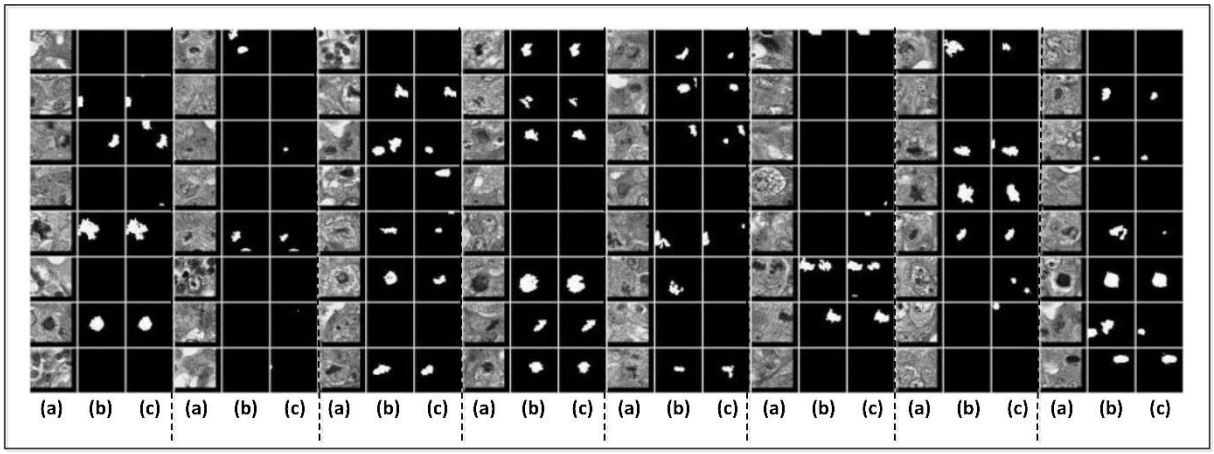



**Figura 3.2 – Resultados da validação do melhor indivíduo.**
**Fonte: Adaptado de (HARDING et al., 2012).**

As imagens da Figura 3.2 estão classificadas da seguinte forma: (a) coluna com as imagens de entrada; (b) coluna com o resultado esperado; (c) coluna com as

imagens das classificações previstas. Segundo Harding (HARDING et al., 2012), o CGP-IP pode trabalhar com as operações de processamento de imagens conhecidas, ao qual tais operações fazem com que os programas gerados tanto sejam mais simples e legíveis e, em comparação com outras abordagens, tais como redes neurais, esta pode tornar a GP mais atrativa para a indústria.

## 3.3 Classificação de Terreno em Marte (LEITNER et al., 2012)

Para demonstrar essa classificação usando CGP-IP foi utilizada uma imagem panorâmica capturada por um veículo espacial não tripulado *Spirit* (MER-A – *Mars Exploration Rover*), o que é considerado o primeiro uso conhecido da técnica CGP para um problema deste tipo. Assim, neste experimento também é utilizada a biblioteca OpenCV, aliada a outras ferramentas de processamento de imagens como: segmentação, que é o processo de separação do plano de fundo da imagem e poda, para otimizar o código gerado por tal experimento.

### 3.3.1 Treinamento do CGP-IP

A seleção do conjunto de treinamento (imagens segmentadas) é de extrema importância para encontrar as melhores soluções. Portanto, se o conjunto é escolhido corretamente, os programas resultantes são filtros muito robustos. Na Figura 3.3 são demonstradas as imagens utilizadas na evolução.

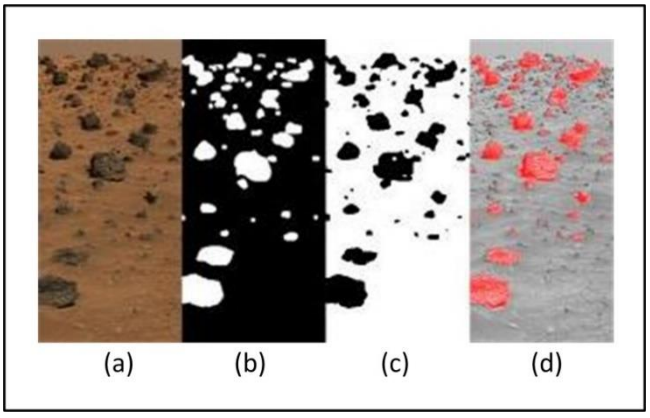



**Figura 3.3 – Uma breve descrição da CGP-IP.**
**Fonte: Adaptado de (LEITNER et al., 2012).**

A Figura 3.3 é composta por quatro imagens: (a) imagem original (MER-A); (b) imagem da segmentação das rochas (classificação feita à mão por um especialista); (c) é a imagem de saída gerada pela técnica CGP, que representa um filtro gerado pelo sistema CGP-IP; (d) uma imagem sobreposta com um nível de cinza (apenas para visualização). Em outras palavras, a partir da imagem entrada juntamente com sua segmentação, é desenvolvido um programa para realizar a mesma segmentação. A saída deste programa é a imagem invertida da segmentação, onde para inspeção visual a mesma foi sobreposta. Em uma seção da imagem panorâmica todas as rochas são rotuladas e são usadas na aprendizagem do programa CGP-IP para esta tarefa. Segundo J. Leitner (LEITNER et al., 2012), a busca evolucionária encontra um resultado válido depois de apenas 8.000 avaliações. O tempo de execução do programa para classificar o conteúdo da imagem de treinamento (347 × 871 *pixels*) é em torno de 400ms. Em outra versão da imagem panorâmica (8939 × 2308), a classificação leva cerca de 30 segundos.  A Figura 3.4 é uma imagem das rochas detectadas pelo sistema CGP-IP a partir de uma parte da imagem panorâmica.

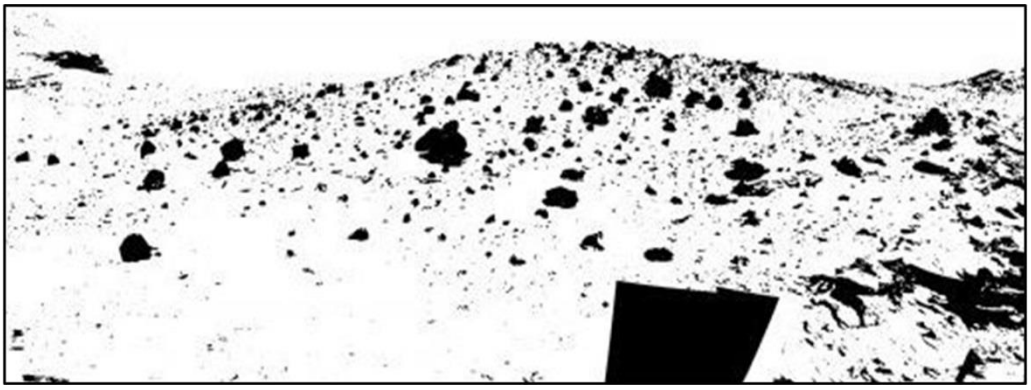


**Figura 3.4 – Corte binário de uma parte da imagem panorâmica de Marte.**
**Fonte: Adaptado de (LEITNER et al., 2012).**

### 3.3.2 Detectando Rochas Específicas

Diferentemente do experimento onde o sistema deveria detectar todas as rochas acima de um determinado tamanho, este experimento apresentado a seguir é para detecção de rochas específicas. Em uma primeira etapa, a rocha ou várias pedras de interesse são marcadas em uma imagem. A seção foi cortada a partir da imagem panorâmica original e redimensionada para a resolução de 539 × 471 *pixels*.

A Figura 3.5 está dividida em duas imagens: (a) uma parte da imagem panorâmica mostrando a rocha de interesse; (b) a segmentação feita manualmente por um especialista que é usada para o treinamento CGP-IP. A rocha escolhida tem um aspecto visual diferente se comparada com a maioria (porosa e rocha vulcânica).

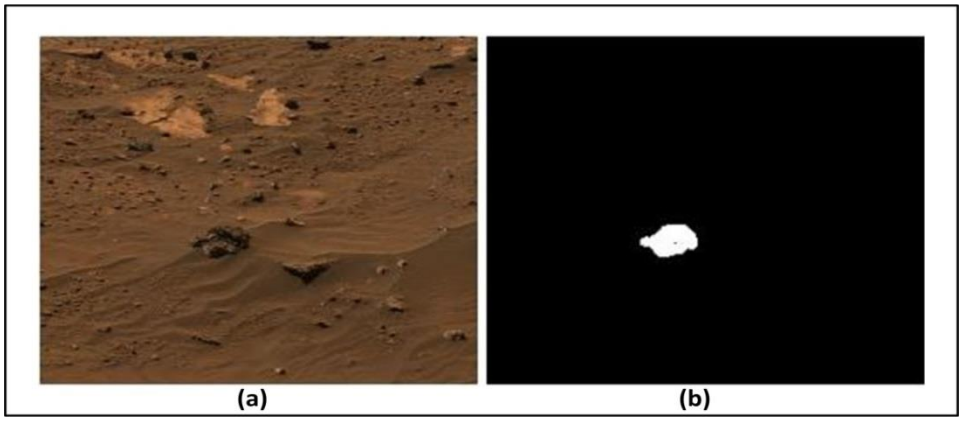



**Figura 3.5 – Imagem da rocha específica de Marte para reconhecimento.**
**Fonte: Adaptado de (LEITNER et al., 2012).**

Outros experimentos para reconhecimento de terrenos foram feitos utilizando o CGP-IP e, segundo J. Leitner (LEITNER et al., 2012), depois de uma curta fase de aprendizagem o sistema mostrou ser consideravelmente mais rápido do que as abordagens anteriores. Estas experiências mostram como o sistema CGP-IP pode ser aplicado para auxiliar a navegação autónoma. Tentando detectar todas as rochas (maiores que um determinado tamanho) para calcular um mapa transversal simples.

## 3.4 CGP para Tarefas de Processamento de Imagens (MONTES; WYATT, 2003)

Para efetuar esse tipo de tarefa o sistema necessita da aplicação de uma série de operadores de processamento de imagens conhecidos, como dilatação, erosão, etc., e geralmente conta com as habilidades e competências dos especialistas no assunto. Como o objetivo deste experimento é encontrar o centro do objeto presente na imagem, o melhor resultado é encontrado quando as coordenadas *(x, y)* têm seus

valores o mais próximo possível uns dos outros. Portanto, a entrada para o programa consiste no conjunto de valores dos *pixels* e a saída do programa consiste na localização do centro do objeto desejado (*x, y)* sem nenhuma informação pré-definida antes da inicialização do sistema. Na Figura 3.6 são ilustrados os objetos do conjunto de treinamento.

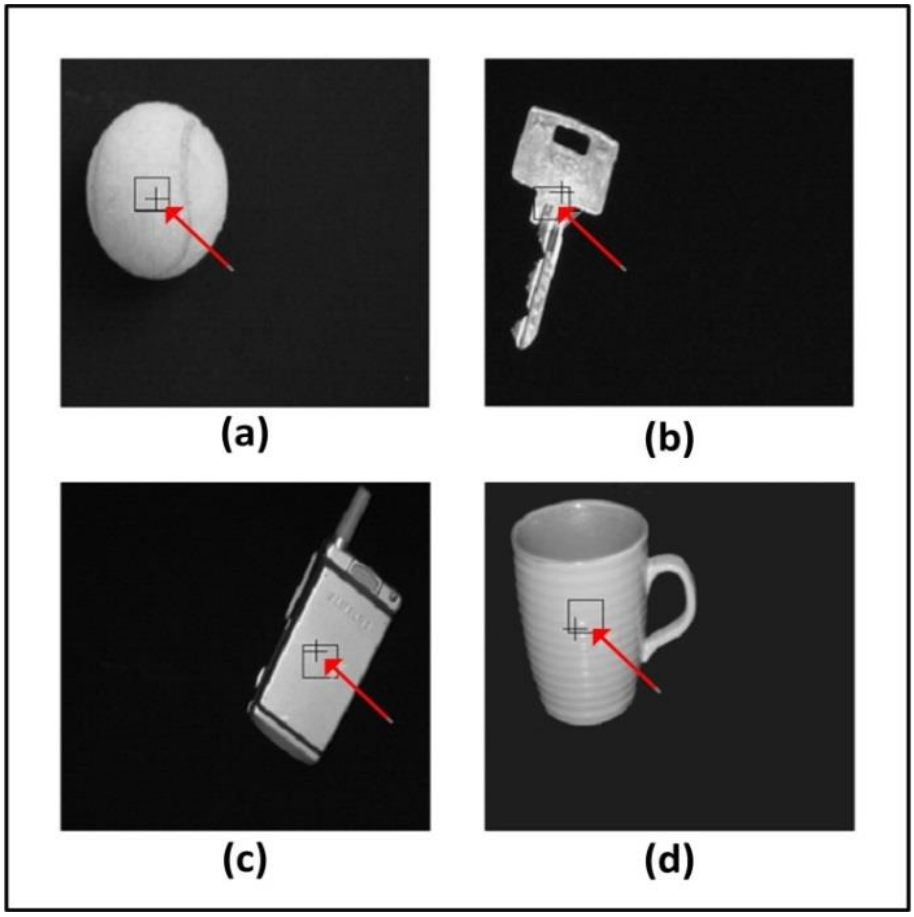



**Figura 3.6 – Conjunto de treinamento.**
**Fonte: Adaptado de (MONTES; WYATT, 2003).**

A Figura 3.6 está organizada da seguinte forma: (a) imagem de uma bola de tênis; (b) imagem de uma chave; (c) imagem de um celular; e (d) imagem de uma caneca. As imagens (estáticas) de entrada utilizadas no experimento como conjunto de treinamento possuem resolução (256×256) com 256 níveis de cinza. A seta (vermelha) mostra duas características deste experimento: (quadrado) que indica a coordenada fornecida pelo usuário; (cruz) que indica as coordenadas encontradas pelo sistema.

### 3.4.1 Resultados dos Experimentos

Para obter os resultados ilustrados da Figura 3.6 foram utilizados dois algoritmos evolucionários: (1) algoritmo com uma variação do operador de recombinação, mais conhecido como “cruzamento uniforme” (escolha dos descendentes aleatoriamente a partir de cinco pais); (2) algoritmo com uma forma simples de estratégia evolucionária (mutação de 1+ʎ indivíduos da população), ou seja, sem a presença do operador de recombinação. Portanto, o primeiro algoritmo apresentou nos testes valores onde 24% de todas as respostas têm seus acertos fora do objeto, 76 % dentro do objeto e 31% marcaram com uma “cruz” área delimitada pelo “quadrado”. O segundo algoritmo obteve 100% de acertos dentro do objeto, sendo 60% deles na área delimitada com o “quadrado”. Assim, segundo Montes e Wyatt (MONTES; WYATT, 2003), o uso da CGP foi bem sucedido na realização das tarefas de localização de objetos, oferecendo uma boa base para os experimentos realizados nesse trabalho.

## 3.5 CGP para Reconhecimento de Caracteres Manuscritos (REHMAN; KHAN, 2011)

O modelo proposto por Rehman e Khan (REHMAN; KHAN, 2011) contém o pré-processamento mínimo para um genótipo CGP. O mesmo é implementado como um algoritmo para evoluir um *hardware* específico para o reconhecimento dos caracteres. Na Figura 3.7 são mostrados a representação de um genótipo CGP e seu respectivo fenótipo.

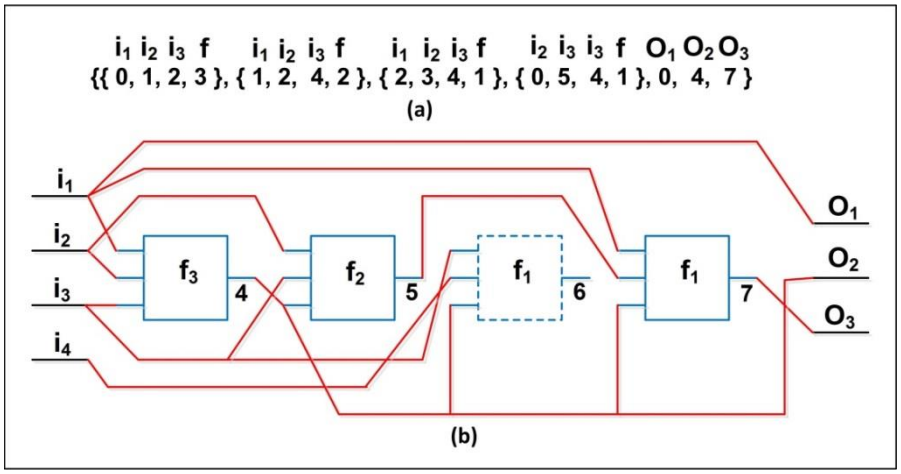



**Figura 3.7 – Genótipo CGP e seu fenótipo.**
**Fonte: Adaptado de (REHMA; KHAN, 2011).**

A Figura 3.7 está dividida em duas imagens: (a) genótipo configurado para um sistema de duas entradas e três saídas, com quatro nós com três entradas de dados, um conjunto de quatro funções e três saídas; (b) Fenótipo obtido a partir do genótipo (a). O nó de número marcado por linhas tracejadas é inativo e não é usado para calcular a saída. Além disso, cada nó pode receber a entrada de qualquer das entradas do sistema ou a partir da saída dos nós anteriores. Para testar o modelo proposto, as imagens foram retiradas da base de dados manuscritos (*MNIST – Mixed National Institute of Standards and Technology Database)*. A Figura 3.8 mostra um exemplo do conjunto de treinamento utilizado para evolução do circuito.

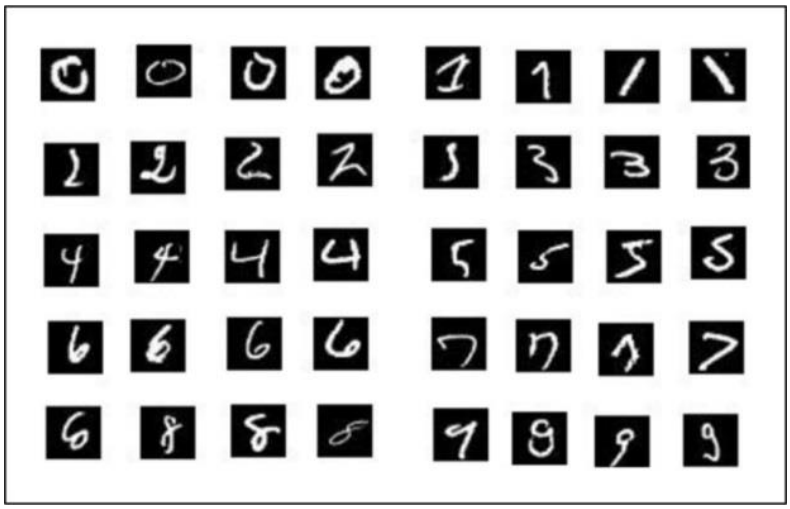


**Figura 3.8 – Conjunto de treinamento com caracteres, com níveis de variação.**
**Fonte: Adaptado de (REHMA; KHAN, 2011).**

### 3.5.1 Experimentos e Resultados

Foram feitos dois tipos diferentes de experimentos. O primeiro experimento possui as seguintes especificações: três entradas para nós; estratégia evolucionária (1+4); quatro funções para o multiplexador de 1*bit*; mesmo número de nós do segundo experimento; número de entradas (matriz de 784 *bits*); número de saídas binárias de quatro *bits*. Na Tabela 3.3 são mostradas as comparações entre os resultados (diferentes sequências de saída) do primeiro experimento.

**Tabela 3.2 – Comparação dos resultados do primeiro experimento**
**Fonte: Adaptado de (REHMA; KHAN, 2011).**

| Nº de nós | Nós Ativos | (Erro %) - Treinamento | (Erro %) - teste | Formato da Saída |
|---|---|---|---|---|
| 1000 | 37 | 12,7 | 21,25 | 4 *bits* |
| 1000 | 46 | 6,5 | 21,1 | ASCII - 8 *bits* |
| 1000 | 45 | 3,72 | 38,3 | Imagem - 25*bits* |
| 500 | 51 | 5,8 | 42,25 | Imagem - 25 *bits* |
| 500 | 44 | 6,36 | 39,75 | Imagem - 25 *bits* |

Os resultados obtidos foram determinados após 100 mil gerações. Segundo Rehma e Khan (REHMAN; KHAN, 2011), de acordo com os dados da Tabela 3.3, fica provado que a formação da imagem (erro de treinamento = 3.72) melhora com o aumento no número de bits de saídas, ao mesmo tempo em que o “erro de teste” é mais alto para 25 bits. O que mostra que a sequência de operadores deve ser escolhida em relação à similaridade dos caracteres (números).

O segundo experimento é configurado com as seguintes características: três entradas para nós; estratégia evolucionária (1+9); dezesseis funções para o multiplexador de 1*bit*; mesmo número de nós do primeiro experimento; número de entradas (matriz de 784 *bits*); e número de saídas binárias de quatro *bits*. A Tabela 3.4 mostra a comparações entre os resultados (diferentes sequências de saída) do segundo experimento. Os resultados da Tabela 3.4 mostram que o aumento no número de imagens no conjunto de treinamento provoca alterações no padrão de entrada e consequentemente diminui a eficiência. Por outro lado, o erro percentual de teste é menor, pois o sistema possui mais parâmetros para comparação.

**Tabela 3.3 – Comparação dos resultados do segundo experimento.**
**Fonte: Adaptado de (REHMA; KHAN, 2011).**

| Nº de nós | Nº de Imagens | (Erro %) - Treinamento | (Erro %) - teste | Nós Ativos |
|---|---|---|---|---|
| 1000 | 50 | 6.6 | 22 | 36 |
| 2000 | 50 | 8.8 | 25.7 | 38 |
| 5000 | 50 | 10.2 | 24.7 | 94 |
| 1000 | 100 | 8.4 | 33.3 | 67 |
| 2000 | 100 | 5.8 | 32.9 | 62 |
| 5000 | 100 | 11 | 30.4 | 85 |
| 1000 | 1000 | 17.8 | 21.8 | 32 |
| 2000 | 1000 | 15.6 | 20.5 | 23 |
| 5000 | 1000 | 18.5 | 22.7 | 53 |

Segundo Rehma e Khan (REHMAN; KHAN, 2011), tal trabalho conseguiu obter um circuito usando o algoritmo de CGP, que pode identificar caracteres escritos à mão com uma eficiência de aproximadamente 80% usando o mínimo de pré-processamento, e que pode ser facilmente implementado em FPGA.

## 3.6 Arquitetura Reconfigurável Virtual para *Hardware* Evolutivo (WANG et al., 2008)

O artigo publicado contendo tal arquitetura foi utilizado como base para o desenvolvimento deste projeto de mestrado, pois o mesmo utiliza na evolução das arquiteturas em *hardware* as regras da CGP, buscando fazer o reconhecimento de caracteres e filtragem de imagens, ambos de forma automática. Diante desta solução, a proposta de desenvolvimento do sistema SHAIP-CGP em nível de *software* busca diminuir o trabalho de compreensão do funcionamento e da escolha dos operadores para as possíveis arquiteturas a serem modeladas, baseadas em processamento de imagens binárias. Assim, o motivo principal da pesquisa do VRA é a concepção de um sistema evolutivo geral, flexível, com capacidade computacional para conseguir realizar evoluções intrínsecas.

### 3.6.1 Implementação do VRA

Toda a organização do VRA foi implementada em uma placa PCI "*Celoxica RC1000*". Esta inclui uma FPGA "*Xilinx Virtex XCV2000E"* com 8MB de memória (SRAM), que está diretamente ligada a FPGA através de quatro bancos de memória de 32 *bits*. Existem dois métodos de transferência de dados do PC para a FPGA: (1) dois caminhos de 8 *bits* unidirecionais chamados de controle e *status* da porta, (2) quatro bancos de memória de 32 *bits* (0-3). Esta organização do VRA é dada na Figura 3.9.

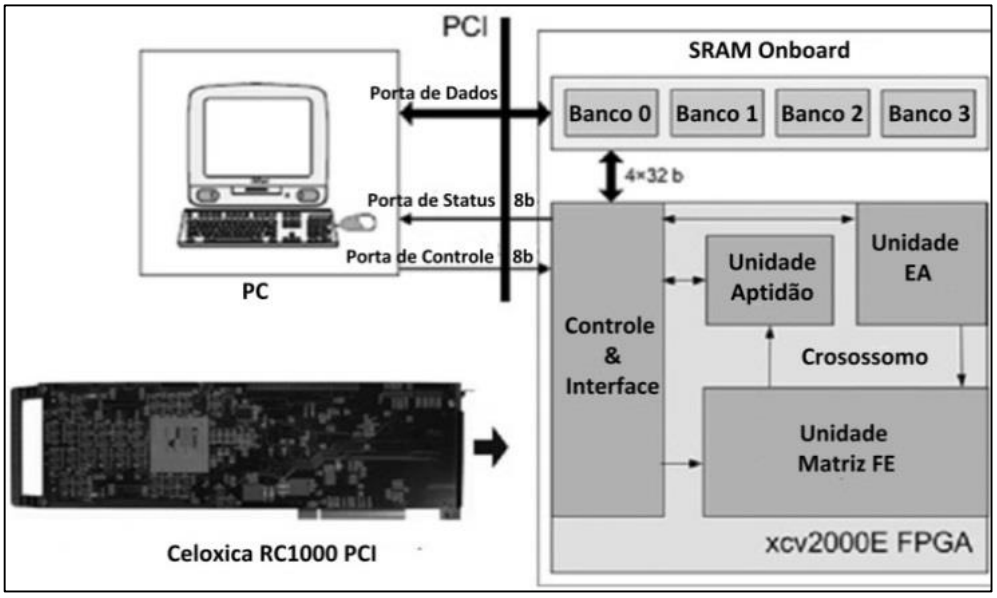



**Figura 3.9 – Organização do VRA na placa Celoxica.**
**Fonte: Adaptado de (WANG et al., 2008).**

Na Figura 3.9 o sistema baseado na evolução do VRA proposto sobre uma FPGA é constituído por quatro componentes: a Unidade da Matriz denominada elemento função (FE - *Function Element*), que é responsável pela evolução com as regras da CGP; a Unidade EA, que determina a estratégia evolucionária utilizada na implementação através do algoritmo evolutivo; a Unidade de Aptidão que faz a avaliação da aptidão dos circuitos candidatos enviados para a unidade da matriz FE; e finalmente a Unidade de Controle & Interface, que é realizada por meio de uma máquina de estados finitos e quatro contadores de endereço.

### 3.6.2 VRA utilizado para reconhecimento de caracteres em nível de porta de comunicação

É importante ressaltar que apesar deste artigo apresentar mais de um experimento, nesta seção são descritas as características e os resultados encontrados na utilização do VRA para o reconhecimento de caracteres (letras do alfabeto). Na Figura 3.10 é dado um esboço do sistema de reconhecimento de caracteres.


**Figura 3.10 – Organização do VRA para reconhecimento de caracteres.**
**Fonte: Adaptado de (WANG et al., 2008).**

Este sistema evolucionário de reconhecimento de caracteres foi projetado para identificar 16 diferentes padrões binários (letras de A até P), conforme ilustrado na Figura 3.10a. Cada entrada (caractere) tem resolução de (5×6) *pixels*, com valor de 0 ou 1. Assim para o reconhecimento dos mesmos, o sistema evolutivo de reconhecimento de caracteres (veja, Figura 3.10b) possui uma porta de entrada de 30 *bits*. Portanto, cada entrada corresponde a um *pixel* em um caractere, ao mesmo tempo em que cada uma das 16 saídas corresponde a um caractere do conjunto de treinamento.

#### 3.6.2.1 Resultados do Experimento

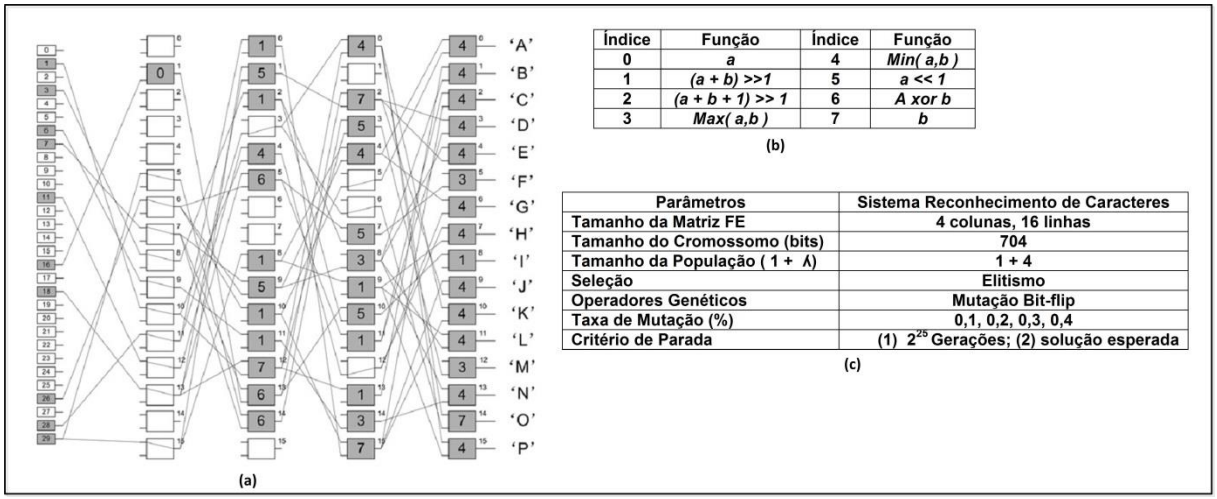
Para esse experimento envolvendo reconhecimento de caracteres foram realizadas 100 evoluções para cada taxa de mutação, sempre gerando a população inicial aleatoriamente. Para estes experimentos foram utilizados dois tipos de equipamento: (1) uma placa FPGA, com 33 MHZ de *clock*, que apresentou um tempo de evolução de 0,019 segundos e um *Speed-up*$_{time}$ de 169; (2) um PC Pentium IV com

2000MHz de *clock,* com uma placa *Celoxica RC 1000 PCI,* e que apresentou um tempo de evolução de 3,204 segundos e um *Speed-up*$_{time}$ de 115. A Tabela 3.4 contém os dados relativos à média e ao desvio-padrão do experimento de reconhecimento de caracteres.

**Tabela 3.4 – Resultados obtidos nas evoluções de reconhecimento de caracteres.**
**Fonte: Adaptado de (WANG et al., 2008).**

| Taxa de mutação | Tempo de Evolução (s) | Número de Gerações | | | |
|---|---|---|---|---|---|
| | | Média | Desv. Padrão | min | max |
| 0,1 | 2,236 | 1 199 599 | 1 709 702 | 145 841 | 13 597 095 |
| 0,2 | 1,108 | 571 557 | 556 557 | 79 949 | 2 978 566 |
| 0,4 | 4,246 | 2 189 172 | 2 331 161 | 141 678 | 12 317 617 |
| 0,8 | 3,483 | 1 795 993 | 1 348 949 | 288 986 | 7 454 084 |

A Tabela 3.4 resume os experimentos e, segundo Wang (WANG et al., 2008), o sistema evolutivo tem um melhor desempenho quando a taxa de mutação é de 0,2%. Um exemplo do sistema de reconhecimento de caracteres evoluído é mostrado na Figura 3.11.




| Índice | Função | Índice | Função |
|---|---|---|---|
| 0 | *a* | 4 | *Min( a,b )* |
| 1 | *(a + b) >>1* | 5 | *a << 1* |
| 2 | *(a + b + 1) >> 1* | 6 | *A xor b* |
| 3 | *Max( a,b )* | 7 | *b* |

| Parâmetros | Sistema Reconhecimento de Caracteres |
|---|---|
| Tamanho da Matriz FE | 4 colunas, 16 linhas |
| Tamanho do Cromossomo (bits) | 704 |
| Tamanho da População ( 1 + λ) | 1 + 4 |
| Seleção | Elitismo |
| Operadores Genéticos | Mutação Bit-flip |
| Taxa de Mutação (%) | 0,1, 0,2, 0,3, 0,4 |
| Critério de Parada | (1) $2^{25}$ Gerações; (2) solução esperada |

**Figura 3.11 – Evolução do sistema na FPGA, funções e parâmetros.**
**Fonte: Adaptado de (WANG et al., 2008).**

A Figura 3.11, está dividida em 3 imagens: (a) a evolução do sistema, onde foram empregadas 41 FE's; (b) tabela com as funções e seus respectivos índices; (c) tabela com os valores dos parâmetros utilizados nesse experimento. Segundo Wang

(WANG et al., 2008), a abordagem utilizando um VRA oferece muitos benefícios, incluindo: (1) na VRA , a matriz FE está diretamente ligada à implementação de *hardware* da AE, colocados na mesma FPGA permitindo que o gargalo de comunicação, que é em muitos momentos lento entre a FPGA e o computador, possa ser melhorado; (2) como o VRA está disponível ao nível do código fonte em linguagem de descrição de *hardware* (HDL – *Hardware Description Language)*, ele pode ser facilmente modificado e sintetizado por várias plataformas; (3) o sistema VRA pode ser concebido exatamente de acordo com as exigências de um determinado problema. Os resultados experimentais mostram que a VRA pode trazer maior capacidade computacional e mais flexibilidade do que a abordagem tradicional para o EHW intrínseco.

## 3.7 Sistema Baseado em GP para a Construção Automática de Filtros de Imagem (PEDRINO et al., 2013)

Conforme citado no início deste capítulo, desse trabalho desenvolvido por Pedrino (PEDRINO et al., 2013) foram utilizados os dados referentes aos resultados obtidos em um de seus experimentos (processamento de imagens binárias) para serem comparados com os resultados obtidos nos estudos de caso do simulador SHAIP-CGP desenvolvido neste trabalho de mestrado. Esse sistema foi implementado com o nome de IFbyGP, o qual foi implementado possibilitando utilizar a representação para o desenvolvimento de um *hardware* evolutivo. Portanto, o sistema permite estender os operadores gerados além do caso binário, proporcionando meios para lidar com imagens em tons de cinza e coloridas. Nesta seção serão descritos apenas os experimentos com imagens binárias.

### 3.7.1 Processamento de Imagens Binárias

O estudo de caso, específico para processamento de imagens binárias do IFbyGP, utilizou quatro padrões de objetos (quadrados, círculos, estrelas e anéis), que podem ser identificados da seguinte forma na Figura 3.12: (a) TS1, imagens com

resolução de 209×169; (b) TS2, com imagens com resolução de 65×67 e TS3, com as imagens com resolução de 31×30.

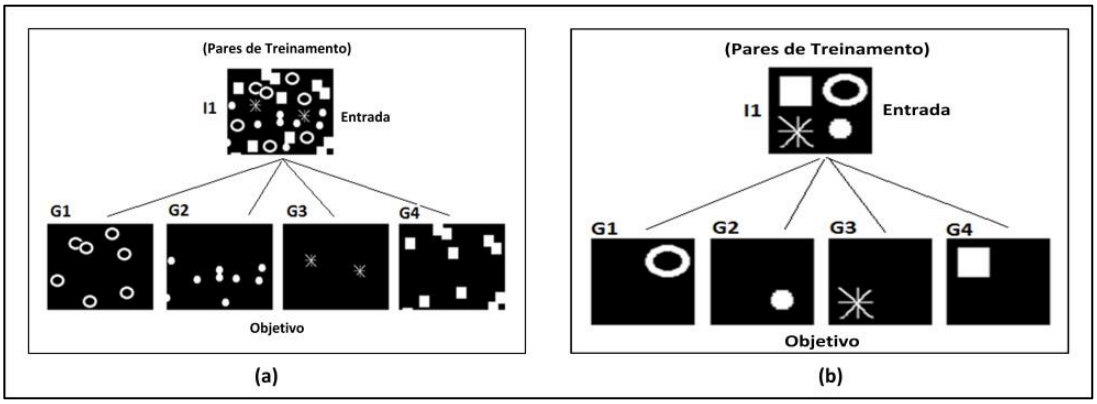



**Figura 3.12 – Conjuntos de treinamento IFbyGP.**
**Fonte: Adaptado de (PEDRINO et al., 2013).**

O processo de formação foi representado por 108 pares (I-G), tendo em conta as três resoluções citadas acima, e o tamanho do cromossomo variou de acordo com um conjunto ($S \in \{8, 16, 32\}$). A Tabela 3.5 mostra a porcentagem de erros (%) para as diferentes amostras de objetos, com as variações no tamanho do cromossomo (S) e das funções ($f_1$, $f_2$, $f_3$).

**Tabela 3.5 – Porcentagem de erros para as diferentes formas.**
**Fonte: Adaptado de (PEDRINO et al., 2013).**

| Amostras | S=8 $f_1$ | S=8 $f_1$ | S=8 $f_1$ | S=16 $f_2$ | S=16 $f_2$ | S=16 $f_2$ | S=32 $f_3$ | S=32 $f_3$ | S=32 $f_3$ |
|---|---|---|---|---|---|---|---|---|---|
| TS1 - Quadrado | 3,53 | 14,51 | 4,82 | 1,48 | 14,42 | 4,7 | 1,81 | 7,01 | 2,7 |
| TS1 - Anel | 5,86 | 15,69 | 4,48 | 5,03 | 25,57 | 4,4 | 5,71 | 14,81 | 4,91 |
| TS1 - Estrela | 0,005 | 13,00 | 4,84 | 0,1 | 6,92 | 6,85 | 0,005 | 14,26 | 0,361 |
| TS1 - Círculo | 4,00 | 58,17 | 2,95 | 3,2 | 54,05 | 2,96 | 2,83 | 51,71 | 2,87 |
| TS2 - Quadrado | 3,86 | 16,47 | 4,79 | 3,83 | 15,03 | 2,33 | 1,97 | 16,3 | 1,93 |
| TS2 - Anel | 3,08 | 36,49 | 3,33 | 2,8 | 14,36 | 3,47 | 2,25 | 20,15 | 3,48 |
| TS2 - Estrela | 0,00 | 8,32 | 0,20 | 0,00 | 0,00 | 0,68 | 0,30 | 0,00 | 0,22 |
| TS2 - Círculo | 4,00 | 50,89 | 2,86 | 3,4 | 51,08 | 2,76 | 2,89 | 28,3 | 2,37 |
| TS3 - Quadrado | 3,66 | 1,13 | 0,17 | 0,54 | 3,43 | 0,67 | 0,54 | 1,70 | 0,34 |
| TS3 - Anel | 3,66 | 26,15 | 4,27 | 2,04 | 23,09 | 3,98 | 4,30 | 28,21 | 4,64 |
| TS3 - Estrela | 0,86 | 6,67 | 0,55 | 0,00 | 7,36 | 0,47 | 0,00 | 27,73 | 1,26 |
| TS3 - Círculo | 1,83 | 51,3 | 2,45 | 0,65 | 18,89 | 1,47 | 0,86 | 1,45 | 1,85 |

A Figura 3.13 ilustra a saída do IFbyGP configurado para detectar o padrão “estrela”.

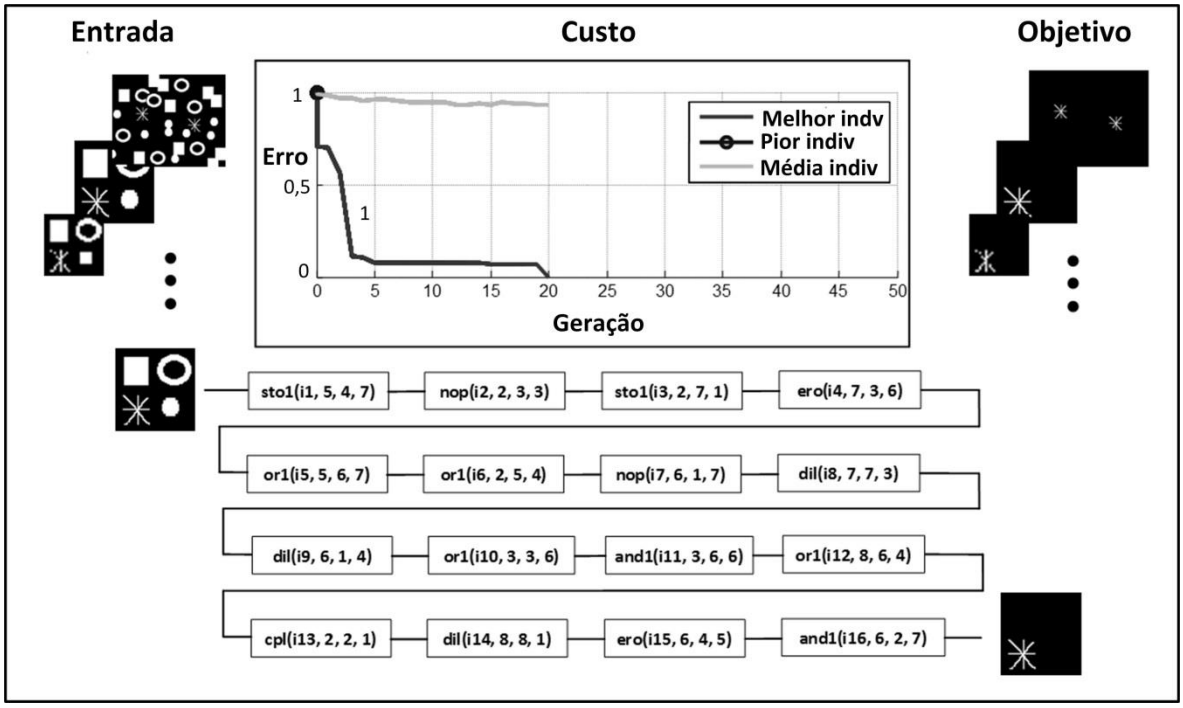



**Figura 3.13 – Exemplo de uma saída do processo do IFbyGP.**
**Fonte: Adaptado de (PEDRINO et al., 2013).**

Segundo Pedrino (PEDRINO et al., 2013), é possível observar na Figura 3.13 que o padrão gerado, resultante da aplicação da sequência de operadores encontrados pelos IFbyGP a uma das entradas, é uma combinação perfeita. O gráfico referente ao custo mostra que o algoritmo precisou de 20 gerações para obter um erro zero. Segundo Pedrino (PEDRINO et al., 2013), o algoritmo baseado em GP pode ser usado em aplicações relacionadas com a segmentação de imagens e reconhecimento de padrões. Também o método desenvolvido pode ser usado em outras aplicações que envolvam o processamento de imagens linear (reconhecimento de notas musicais, processamento de imagens em nível de cinza, reconhecimento de laranjas em um pomar).

## 3.8 Considerações Finais

Além de abordagens destinadas aos especialistas humanos, um grande progresso tem sido feito através de técnicas de aprendizado de máquina para executar a classificação de imagens. Assim, todos os artigos apresentados neste capítulo tiveram alguma importância no desenvolvimento deste projeto de mestrado, no qual podemos destacar a escolha correta dos operadores (morfológico, lógicos, etc.), a determinação do tamanho do cromossomo, a determinação da estratégia evolucionária, a resolução correta para as imagens, a função de aptidão, a taxa de mutação, o número médio de gerações, entre outras.

# Capítulo 4

# *PROPOSTA DO TRABALHO*

*Nesta seção, são apresentados os principais aspectos e funcionalidades do simulador SHAIP-CGP, com especial ênfase no seu algoritmo principal baseado em CGP e seus procedimentos relacionados. Na subseção 4.1.1, é descrito o procedimento geral para construção do conjunto de treinamento utilizado nos experimentos que serão relatados no próximo capítulo. Na subseção 4.1.2, são ilustradas as entradas do SHAIP-CGP. Na subseção 4.1.3, são apresentados os operadores lógicos e morfológicos. Na subseção 4.1.4, é descrita a estratégia evolucionária do SHAIP-CGP. Na subseção 4.1.5, é ilustrado o bloco CGP. Na subseção 4.1.6, são descritas as variáveis do simulador. Na subseção 4.1.7, é apresentada a função de custo. E finalmente na seção 4.1.8, é apresentada a implementação do SHAIP-CGP.*

## 4.1 Simulador de Arquitetura para Processamento de Imagens Usando Programação Genética Cartesiana

O trabalho proposto é um simulador baseado em CGP e foi desenvolvido para produzir automaticamente um programa para filtragens de imagens binárias. A primeira etapa para a utilização do SHAIP-CGP é preparar um conjunto de treinamento (composto de imagens binárias) adequado, utilizado pelo sistema como uma de suas entradas, auxiliando-se, assim, o simulador a inferir um programa capaz o suficiente para filtrar tipos específicos de imagens.

### 4.1.1 Procedimento para Criar o Conjunto de Treinamento

O procedimento geral para a criação de um conjunto de treinamento para o sistema consiste, primeiramente, na aquisição de uma imagem original binária (O), sem a presença de ruído. Baseado nessa imagem, um especialista humano utiliza ferramentas computacionais para a criação de quatro novas imagens, conhecidas como: Alvo_$O_1$, Alvo_$O_2$, Alvo_$O_3$, e Alvo_$O_4$, das quais cada uma, contém uma característica especial de interesse contida na imagem original (O), como mostrado na Figura 4.1.

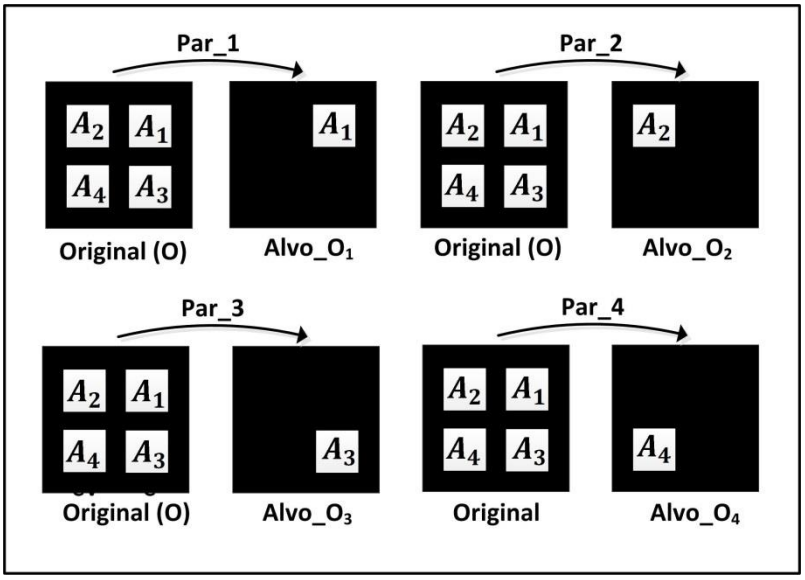



**Figura 4.1 – Conjunto de treinamento do SHAIP-CGP (quatro pares de imagens).**

O conjunto de treino CT, é então, criado tendo quatro pares de imagens: CT = {(O, Alvo_$O_1$), (O, Alvo_$O_2$), (O, Alvo_$O_3$), (O, Alvo_$O_4$)}. A primeira imagem de cada par alimenta o processo de geração do cromossomo CGP, e a segunda alimenta a função de custo utilizada no processo de avaliação da aptidão, que é uma parte inerente da estratégia evolucionária empregada no SHAIP-CGP.

### 4.1.2 Entradas do SHAIP-CGP

Como citado anteriormente, por ser um simulador baseado em CGP e que vai trabalhar especificamente com morfologia matemática, faz-se necessária a preparação das entradas (neste caso imagens binárias e elementos estruturantes).

Assim, o simulador vai ter como dados de entrada as imagens (retiradas do conjunto de treinamento) e os elementos estruturantes, ambos ilustrados na Figura 4.2.

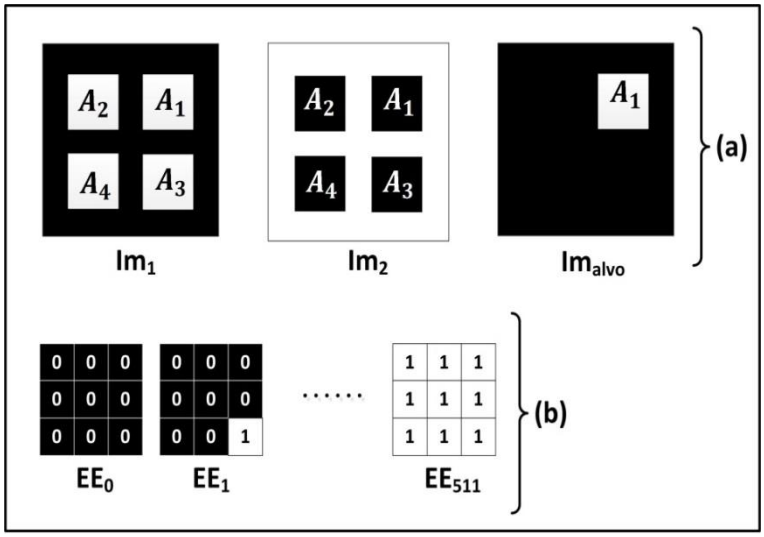



**Figura 4.2 – Ilustração das entradas do SHAIP-CGP.**

A Figura 4.2 está dividida da seguinte forma: (a) conjunto de imagens, no qual $Im_1$ é uma imagem binária (contendo quatro formas diferentes de objetos), retirada do conjunto de treinamento, $Im_2$ é a imagem $Im_1$ com os valores dos *pixels* invertidos, ou seja, onde o valor é "0" passa a ser "1" e vice-versa. A imagem $Im_{alvo}$ é uma das imagens-alvo (objetivo), também retirada do conjunto de treinamento contendo apenas um dos objetos pertencentes a $Im_1$. Como mencionado anteriormente, as imagens são binárias, onde os "1's" representam os pixels brancos e os "0's" representam os pixels pretos; (b) conjunto de elementos estruturantes com resolução de 3×3 *pixels* e, portanto, com $2^9$ (512) diferentes possibilidades de formas.

### 4.1.3 Operadores

Além do conjunto de treinamento, um simulador baseado em CGP necessita de funções (operadores). Este simulador de arquitetura possui alguns tipos específicos de funções que são escolhidas para codificar os genes que formam o cromossomo CGP. É importante ressaltar que, nas chamadas funções genes, o tipo de operador utilizado depende da localização do gene dentro do *grid;* por exemplo, um operador lógico "AND" necessita de duas imagens com o mesmo tamanho, já o operador morfológico de dilatação utiliza uma imagem maior e outra menor (elemento

estruturante). Na Tabela 4.1, é apresentada uma lista contendo oito operadores utilizados na implementação do SHAIP-CGP, sendo que eles foram escolhidos através de testes preliminares de compatibilidade feitos por um especialista humano.

**Tabela 4.1 – Operadores utilizados na implementação do SHAIP-CGP.**

| Operador | Nº Inteiro Associado | Descrição |
|---|---|---|
| dilatação | 0 | Operador Morfológico de Dilatação |
| erosão | 1 | Operador Morfológico de Erosão |
| bypass | 2 | Operador de identidade (relacionado à conexão de entrada $C_{E1}$ dos genes (veja Figura 4.3b)). |
| and | 3 | Operador Lógico AND |
| or | 4 | Operador Lógico OR |
| nor | 5 | Operador Lógico NOR |
| bypass | 6 | Operador de identidade (relacionado à conexão de entrada $C_{E1}$ dos genes (veja Figura 4.3b)). |
| bypass | 7 | Operador de identidade (relacionado à conexão de entrada $C_{E2}$ dos genes (veja Figura 4.3b)). |

Uma lista de oito operadores é utilizada na implementação do SHAIP-CGP. Entre as oito, três delas são de operadores de identidade *bypass:* (1) *bypass* codificado com o número inteiro 2. Este operador, simplesmente, permite que o dado (imagem) presente na entrada do gene seja transferido para a sua saída, sem qualquer alteração; (2) os operadores codificados com os números inteiros 6 e 7 atuam sobre $C_{E1}$ e $C_{E2}$, respectivamente. Esta notação foi adotada devido às regras CGP, que estão descritas na seção 4.1.5. A associação dos operadores utilizados pelo SHAIP-CGP pode gerar outros operadores morfológicos primitivos, como, por exemplo, os de fechamento e abertura.

### 4.1.4 Estratégia Evolucionária

Uma variante muito simples (1+ʎ) de um algoritmo evolucionário é utilizada na implementação do SHAIP-CGP, no qual ʎ é o número de cromossomos (filhos) pertencentes a cada nova geração. Portanto, essa estratégia evolucionária é para evoluir os cromossomos de uma geração para a seguinte, na qual o cromossomo-pai está inalterado e ʎ filhos são produzidos por mutação a partir dele. No Procedimento 4.1 (MILLER, 2011), são descritas as etapas do algoritmo evolucionário que determina a estratégia evolucionária do SHAIP-CGP.

**Procedimento 4.1 - Estratégia Evolucionária (1+ʎ)**

1: **para todo** $i$ tal que $0 \leq i < (1 + \lambda)$ **faça**
2: *Gerar indivíduos (cromossomo) aleatoriamente i*
3: **fim para**
4: *Selecionar o indivíduo com melhor aptidão, para ser promovido a pai.*
5: **enquanto** uma *solução não é encontrada* ***ou*** *o limite da geração não é alcançado* **faça**
6: **para todo** $i$ tal que $0 \geq i < \lambda$ **faça**
7: $Mutar\ o\ pai\ para\ gerar\ descendência\ i$
8: **fim para**
9: *Gerar o indivíduo com melhor aptidão utilizando a seguinte regra:*
10: **se** *o filho tiver melhor aptidão que o pai* **então**
11: *O filho é escolhido como o mais apto*
12: **senão**
13: *O pai permanece como o mais apto*
14: **fim se**
15: **fim enquanto**

Das etapas descritas no algoritmo evolucionário do SHAIP-CGP, a de número 10 possui uma condição extra que deve ser considerada: o filho que for avaliado com a aptidão igual ou superior à do pai deve ser promovido a pai, para criar a próxima população.

### 4.1.5 Construção do Bloco CGP

Como em todas as implementações baseadas em CGP, o *grid* adotado neste simulador também pode ser abordado como um grafo acíclico direcionado; no entanto, se ele for comparado com a programação genética cartesiana original (MILLER; THOMSON, 2000), sua implementação inclui mais restrições *ad-hoc*, no que diz respeito à comunicação entre os genes pertencentes ao *grid*, pois essa comunicação também determina qual o tipo de dados (imagens de entrada, elementos estruturantes e imagens das conexões de saída dos genes) que os genes devem receber, já que o simulador nesse sentido é “autônomo” e precisa adaptar-se aos diferentes tipos de operadores (Tabela 4.1). Outro motivo que determinou a adoção dessas restrições é o fato de que, para implementações em *hardware,* é necessário que o comprimento do genótipo seja o mais reduzido possível, simplificando assim a geração do fenótipo. Na Figura 4.3, serão ilustradas: (a) arquitetura funcional do bloco CGP (forma geral representada pelo *grid* (entradas, genes e saída), que vai receber as respectivas regras de acordo com as restrições impostas ao SHAIP-CGP); (b) arquitetura genérica (entradas, saída, funções) dos genes localizados entre a primeira e a penúltima coluna do *grid*; (c) arquitetura genérica (entradas, saída, funções) dos genes localizados na última coluna do *grid*.

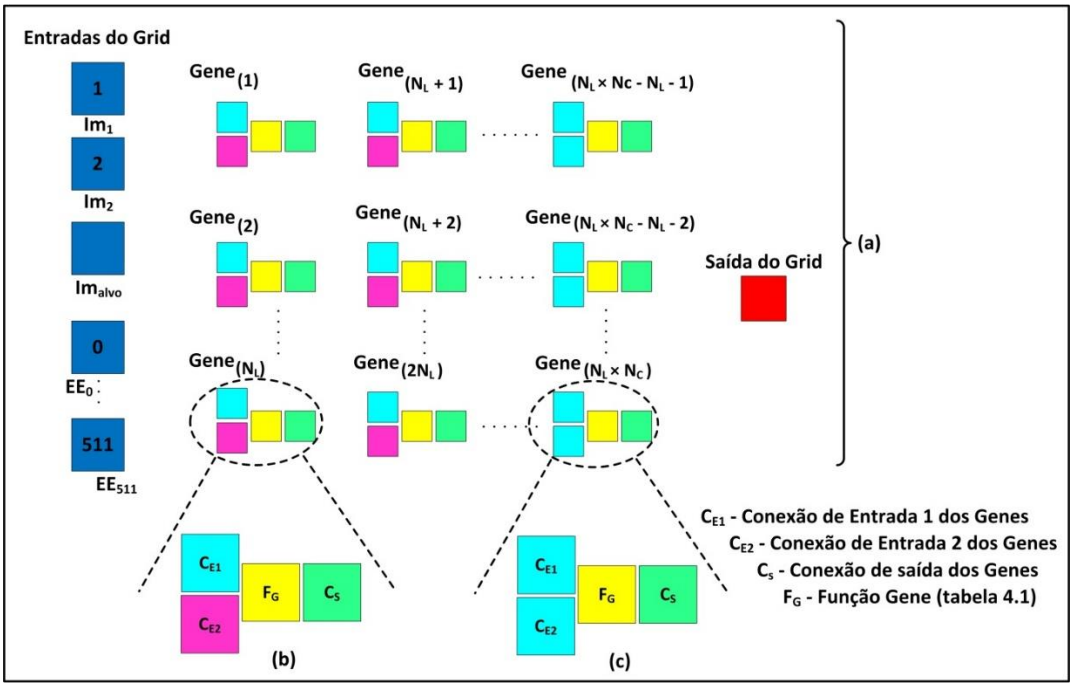



**Figura 4.3 – Arquitetura funcional CGP, com destaque para arquitetura do gene.**

No SHAIP-CGP, a arquitetura funcional (Figura 4.3a) é composta pelo *grid* representado por: um número fixo de genes ($N_{genes}$), determinado automaticamente como o produto das variáveis, número de linhas ($N_{L)}$ pelo número de colunas ($N_C$), ambas informadas pelo usuário antes da execução do programa; um conjunto de entradas (veja, Figura 4.2) e saída do *grid*. A representação CGP deste trabalho pouco se difere da proposta original (MILLER; THOMSON, 2000), em relação à convenção adotada para determinar a sequência de numeração dos genes no *grid*, ou seja, os genes são numerados em sequência e em ordem crescente de cima para baixo e da esquerda para a direita, começando com $Gene_{(1)}$ e terminando com $Gene_{(NL \times NC)}$. Para esses genes, adotou-se uma arquitetura específica (Figura 4.3b e Figura 4.3c) já que o domínio específico desse simulador (processamento de imagens binárias) utiliza operadores lógicos e morfológicos que atuam de forma diferente sobre as imagens binárias e em alguns momentos precisam que elas tenham igualdade de tamanho (número de *pixels*). Portanto, um gene é composto por quatro partes (alelos são as formas alternativas do mesmo de cada parte) que são codificadas por números inteiros e podem receber tipos de dados (imagem, elementos estruturantes) diferentes, de acordo com a coluna em que ele está localizado. Cada parte (alelo) possui as seguintes características: (1) $C_{E1}$ – Conexão de entrada 1 do gene (cor azul), que somente é alimentada por imagens; (2) $C_{E2}$ – Conexão de entrada 2 do gene (cor

rosa), que recebe somente elementos estruturantes; (3) $C_S$ – Conexão de saída do gene (cor verde), que armazena a imagem gerada pela função gene atual; (4) $F_G$ – Função gene (cor amarela), que armazena a codificação dos operadores do SHAIP-CGP (Tabela 4.1).

#### 4.1.5.1 Regra Específica da Primeira Coluna

A conexão de entrada $C_{E1}$ de cada gene somente pode obter seus dados (imagens) da entrada do *grid*, ou seja, um sorteio aleatório entre as imagens $Im_1$ e $Im_2$. A conexão de entrada $C_{E2}$ deve receber (também de forma aleatória) apenas as 512 formas possíveis de elementos estruturantes. A função gene $F_G$, é codificada pelos operadores: dilatação (0), erosão (1) e bypass (2). O sorteio aleatório das conexões de saída $C_S$ deve ser direcionado somente para as entradas $C_{E1}$ dos genes da coluna imediatamente a sua direita (ou seja, à sua frente). Todo processo que envolve essa regra está ilustrado na Figura 4.4.

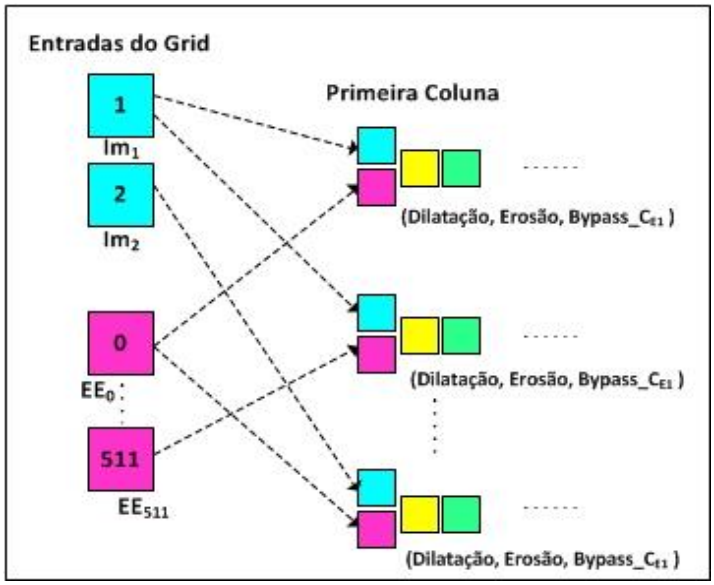



**Figura 4.4 – Regra específica da primeira coluna.**

#### 4.1.5.2 Regra Específica da Segunda até a Penúltima Coluna

A conexão de entrada $C_{E1}$ deve obter seus dados (imagens) do sorteio aleatório das saídas dos genes da coluna imediatamente à sua esquerda (anterior). A conexão de entrada $C_{E2}$ deve receber aleatoriamente o sorteio somente das 512 possíveis formas de elementos estruturantes. A função gene $F_G$, é codificada com os

operadores: dilatação (0), erosão (1) e bypass (2). O sorteio aleatório das conexões de saída $C_S$ deve ser direcionado somente para as entradas $C_{E1}$ dos genes da coluna imediatamente a sua direita (ou seja, à sua frente), com exceção das saídas da penúltima (veja subseção 4.1.5.3). Todo processo que envolve essa regra está ilustrado na Figura 4.5.

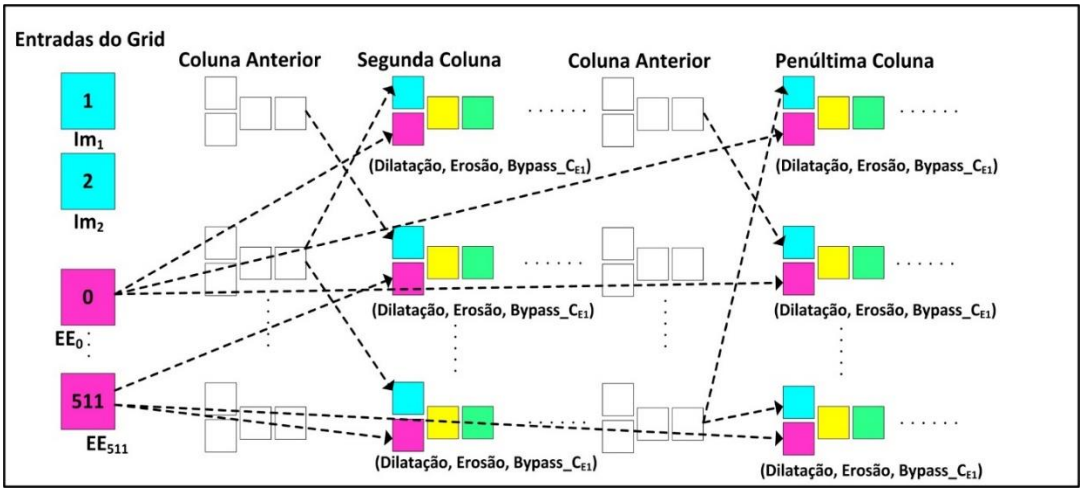



**Figura 4.5 – Regra específica da segunda até a penúltima coluna.**

### 4.1.5.3 Regra Específica da Última Coluna

Para melhor entendimento, faz-se necessário descrever primeiramente a função gene $F_G$, que deve ser codificada com o sorteio aleatório dos seguintes operadores: and (3), or (4), nor (5), bypass (6) e bypass (7). Assim, tanto a conexão de entrada $C_{E1}$ quanto a $C_{E2}$ devem receber seus dados somente do sorteio aleatório das saídas dos genes pertencentes à penúltima coluna, ou seja, $C_{E2}$ (neste caso também ilustrado na cor azul) que nas colunas anteriores recebia somente elementos estruturantes passa a receber somente imagens, devido ao fato de que os operadores lógicos trabalham somente com imagens do mesmo tamanho (mesma resolução). Para garantir que não ocorra uma diferença entre as imagens, foi implementada no SHAIP-CGP uma função de corte para elas; essa função apenas seleciona a imagem maior e, por meio de um “corte”, ajusta-a para a mesma resolução da imagem menor. As saídas $C_S$ são sorteadas aleatoriamente e apenas a de um gene vai ser direcionada para a saída final do *grid*. Todo processo que envolve essa regra está ilustrado na Figura 4.6.

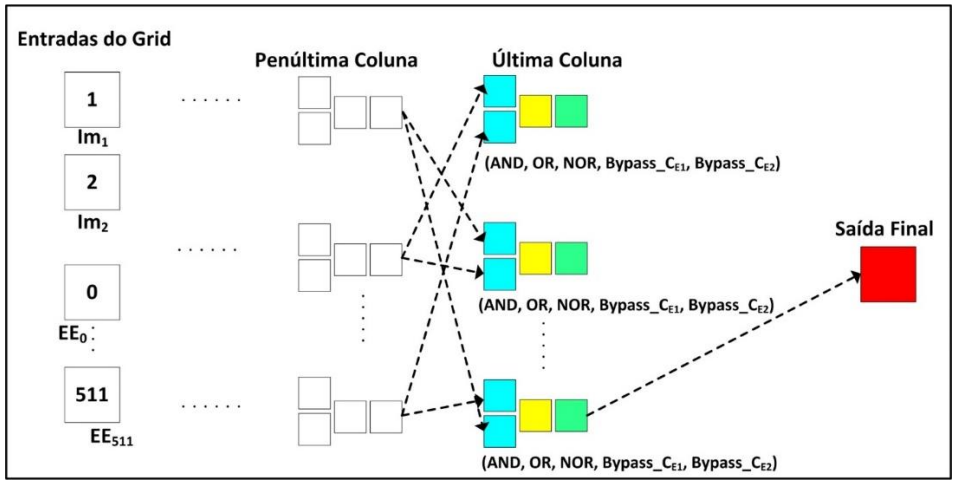



**Figura 4.6 – Regra específica da segunda até a penúltima coluna.**

### 4.1.6 Variáveis do Sistema

Antes de iniciar o treinamento no SHAIP-CGP, é necessária a atribuição, pelo usuário, de valores às variáveis (parâmetros). É importante ressaltar que todas as variáveis são representadas por números inteiros positivos e, por isso, podem ser codificadas e decodificadas de acordo com a exigência do programa. Na Tabela 4.2, são descritas as variáveis de inicialização do simulador.

**Tabela 4.2 – Variáveis envolvidas na inicialização do SHAIP-CGP.**

| Variáveis | Descrição |
|---|---|
| $\lambda$ | Nº de descendentes para criar as novas gerações |
| $N_{crom}$ | Nº de cromossomos para iniciar o sistema (*default* =1) |
| $N_L$ | Nº de linhas (que *define o grid*) |
| $N_C$ | Nº de colunas (que *define o grid*) |
| $N_{genes}$ | Nº de genes do cromossomo |
| $N_{int}$ | Nº de inteiros que representa o *grid* ($N_{int} = N_L \times N_C \times 3$) |
| $N_{im}$ | Nº de imagens de entrada (*default* = 2, $Im_1$ e $Im_2$) |
| $N_{alvo}$ | Nº de imagens alvo (*default* = 1) |
| $N_{EE}$ | Nº de elementos estruturantes (*default* = 512) |
| $N_E$ | Nº de entradas do sistema ($N_{im} + N_{EE}$) |
| $N_S$ | Nº de saídas sistema (*default* = 1) |
| $N_{erro}$ | Nº máximo de erro permitido (critério de parada) |
| $N_{mg}$ | Nº máximo de gerações (critério de parada, *default* = 2000) |
| $M_{taxa}$ | Taxa de mutação (%) |
| $N_{Op}$ | Nº de operadores (*default* = 8) |

É importante evidenciar que, apesar de a variável $N_{alvo}$ armazenar o número de imagens alvo que são associadas a $Im_1$, na entrada do SHAIP-CGP, ela não é somada

à contagem do número de entradas $N_E$ do simulador, devido ao fato de a imagem $Im_{alvo,}$ que inicialmente é carregada no sistema como uma entrada, a mesma somente ser utilizada na avaliação (função de custo), como imagem base para comparação.

### 4.1.7 Função de Custo

A função de custo ($f_c$) implementada no SHAIP-CGP, é específica para trabalhar com imagens binárias. A função é encarregada de avaliar o desempenho de um filtro dado pela melhor sequência de operadores (tendo em conta o mapeamento do cromossomo). A função de custo escolhida calcula o erro médio absoluto (MAE – *Mean Absolute Error*) dado pela equação 4.1 (PEDRINO et al., 2013).

$$d(a,b) = \frac{1}{XY}\sum_{i}^{X}\sum_{j}^{Y}\left|a(i,j) - b(i,j)\right| \quad \textbf{(4.1)}$$

Na equação 4.1, *a* e *b* representam as duas imagens (com o mesmo tamanho) a ser comparadas, em que *a(i, j)* representa um *pixel* na imagem *a*, e *b(i, j)* um pixel correspondente na imagem *b*. Dado o conjunto de treinamento CT, tal como estabelecido na Figura 4.1 no início deste capítulo, a imagem é a mesma que Alvo_$O_i$, e *b* pode ser abordada como o resultado da evolução armazenado na saída do *grid*. Em outras palavras, a função de custo mede o quão se aproxima, em número de *pixels*, a imagem (resultante do processo de filtragem realizada pela sequência de operadores codificados) da imagem filtrada desejada (alvo).

### 4.1.8 Implementação do SHAIP-CGP

Toda a implementação do simulador de arquitetura foi desenvolvida em um ambiente *de programação MATLAB*, na qual a simulação é essencialmente um treinamento do programa cartesiano em um domínio específico (imagens binárias), na qual as regras da CGP garantem que esse treinamento seja autônomo na tomada das decisões durante a evolução. Toda a evolução inserida nesse ambiente de simulação é determinada por cinco etapas implementadas no SHAIP-CGP, que são descritas por um pseudocódigo em alto nível (Figura 4.4) e um fluxograma (Figura 4.5) enfatizando as principais etapas da implementação.

```
procedimento SHAIP
 início
      % definir os valores iniciais de variáveis, elementos estruturantes e funções %
      entradas (Im1,Im2, Imalvo, EE)
      inicializar (λ,Ncrom,NL,NC,Ngenes,Nint,Nim,NEE,NE,NS,Nerro,Nmg,MTAXA,NOp, c_geracoes,entradas)
      gpi ← zeros(Ncrom, Nint) % matriz que armazena a população inicial.
      s ← zeros(1,Ngenes) % matriz que armazena os inteiros que codificam a saída do genes.
      s_gene ← [ ]  % matriz que armazena as conexões de saída dos genes.
      s_final ← [ ] % matriz que armazena a imagens geradas por cada cromossomo.
      erro ← [ ] % matriz que armazena os erros calculados pela função de custo.
      %  geração da população inicial %
      função pop_inicial(gpi,s,NL,NC,NE,Nim,NEE,NOp) % função para gerar a população inicial
      para crom ← 1 até Ncrom faça
          início
             decodificar(s_gene,s_final,Ngenes,crom,gpi,Nim,NEE,NE) % função para decodificar as operações
             avaliar(fc,Im1,s_final,crom) % função para avaliar o resultado (imagem) de cada cromossomo
          fim para
      gpf ← gpi % matriz para armazenar as futuras gerações (1+λ)
      % criando novas gerações
      enquanto (Nerro > 0 or Nmg < 2000) faça
          início
              plot (imagens, gráficos) % mostra a evolução (erro x gerações) e as imagens passo a passo
              c_geracoes ← c_geracoes + 1 % contador de gerações
              nova_gpi ← zeros(λ+1, Nint)
              nova_gpi ← gpf(melhor_crom) % mesma função da matriz gpi, mas usada para novas gerações
              fm(nova_gpi,Mtaxa) % função de mutação
              para novo_crom ← 1 até  λ+1 faça
                 início
                    decodificar(s_gene,s_final,Ngenes,novo_crom,nova_gpi  Nim, NEE, NE)
                    avaliar(fc,Im1,s_final,novo_crom)
                 fim para
        gpf ← nova_gpi
        fim enquanto
      s_final ← (gpf(melhor_crom))  % melhor cromossomo
      plot_gird(s_final) % plota o grid com destaque para os genes ativos.
      retorna(s_final)
 fim procedimento.
```

**Figura 4.7 – Pseudocódigo em alto nível do SHAIP-CGP.**

Como as principais etapas (implementadas como funções no MATLAB) vão ser mais bem descritas seguindo as ilustrações do fluxograma, é necessário descrever algumas das principais linhas de código que são necessárias no contexto geral. Ao iniciar o procedimento SHAIP-CGP, um *script* denominado de *entrada* “carrega” na memória as entradas do sistema (imagens, elementos estruturantes), em seguida são inicializadas juntamente com as matrizes/vetores (*gpi, s, s_gene, s_final e erro*), que são responsáveis por armazenar os dados durante todo o treinamento. A matriz *gpf* recebe a *gpi* (com os cromossomos da primeira geração) contendo o cromossomo “pai” para criar as gerações futuras. A matriz *nova_gpi* tem a mesma função da *gpi,* e é utilizada apenas para diferenciar onde estão os cromossomos gerados pela pop_inicial (primeira geração) dos criados nas gerações futuras. A variável do número máximo de gerações $N_{mg}$ é incrementada, tendo a função de um contador para determinar um dos critérios de parada. Por fim, a variável como *plot(imagens, gráfico)* demonstra na tela do computador as imagens juntamente com o gráfico da evolução (desde o primeiro cromossomo até o considerado como melhor no final). Também é possível plotar o grid. Na Figura 4.5, o fluxograma é apresentado ilustrando as cinco etapas da implementação: (1) inicialização; (2) decodificação; (3) avaliação; (4) estratégia evolucionária, (5) finalização.

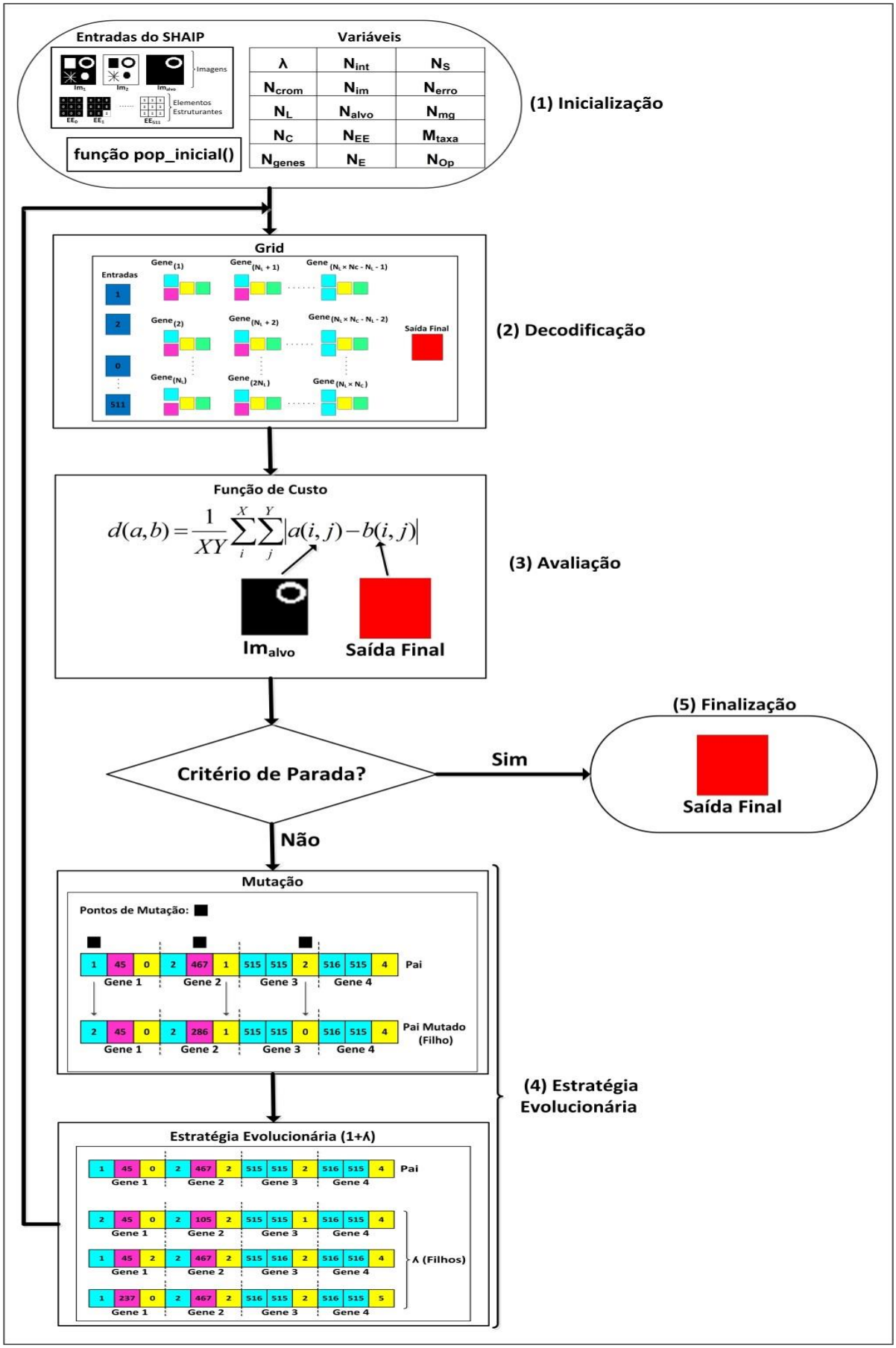



**Figura 4.8 – Fluxograma da implementação do SHAIP-CGP.**

Etapa (1) – inicialização: consiste em adquirir os valores, atribuídos pelo usuário, das variáveis do sistema, conforme listado na Tabela 4.2, juntamente com as imagens e elementos estruturantes (procedimento *inicializar* no pseudocódigo da Figura 4.4). Além disso, parte da *inicialização* é realizada através de uma função denominada "*pop_inicial*", que, de forma aleatória, cria e codifica a população inicial, respeitando o conjunto de restrições apresentadas na subseção 4.1.4. É importante ressaltar que todas as variáveis são representadas por números inteiros positivos e, por esse motivo, podem ser codificadas e decodificadas de acordo com as necessidades do sistema.

Etapa (2) – decodificação: refere-se aos comandos que efetivamente "decodificam" cada uma das partes (alelos) dos genes pertencentes ao *grid,* ou seja, interpreta os códigos até então representados por números inteiros, deixando assim os genes com uma "formatação" compatível com o padrão CGP. Após essa decodificação, é iniciada a evolução propriamente dita do cromossomo, onde as propriedades de cada operador (morfológico, lógico), devidamente armazenado na função gene ($F_G$), são aplicadas sobre os dados pertencentes a cada entrada do gene ($C_{E1}$ e $C_{E2}$) gerando uma imagem que, em seguida, é armazenada na conexão de saída do gene ($C_S$). Assim, com todas as saídas dos genes preenchidas com as respectivas imagens, o SHAIP-CGP separa os genes ativos que foram os responsáveis por gerar a imagem armazenada na saída final. Para identificar esses genes, é necessário traçar um caminho de volta desde a saída final até a primeira coluna. Esse caminho depende da codificação das entradas dos genes, ou seja, se o gene da última coluna (que forneceu a imagem para saída final) tiver suas entradas alimentadas por duas saídas, o caminho é direcionado para os respectivos genes; senão, se forem alimentadas pela saída de um mesmo gene, o caminho é direcionado apenas para ele (gene).

Etapa (3) – avaliação: o processo de avaliação é dividido em dois passos: o primeiro é específico para avaliar a saída de cada cromossomo e é baseado na comparação *pixel* a *pixel* entre duas imagens: a imagem obtida como a saída de um cromossomo (saída final), e a imagem que se refere à imagem-alvo ($Im_{alvo}$), as quais são direcionadas diretamente para a função de custo. A função de custo é empregada pelo processo de avaliação para calcular quantos *pixels* são combinados entre as duas imagens, considerada a resolução da imagem. Esse primeiro passo é executado tanto nos cromossomos da população inicial quanto nas novas gerações, e os erros

são gravados na matriz “erro” (dada desde a etapa (1)); o segundo passo consiste em comparar todos os erros armazenados na matriz “erro” e determinar qual é o menor valor; tendo essa determinação, o cromossomo avaliado com menor erro (mais apto) é promovido como “pai”. Se esse cromossomo possui um erro com o valor igual ao determinado pela variável $N_{erro}$, ou ele mesmo for pertencente à geração igual ao valor da variável $N_{mg,}$ é encaminhado para a etapa (5); caso contrário, segue para a etapa (4).

Etapa (4) – estratégia evolucionária: determina o número de indivíduos da próxima geração, ou seja, a partir do cromossomo “pai”, novas gerações de cromossomos são geradas (1+ʎ), onde o número 1 representa o cromossomo (pai) mutante e ʎ é o número de descendentes a serem gerados. A mutação (subseção 2.2.5.2.2) nesse processo é pontual, na qual, são escolhidos; por um sorteio aleatório, 10% do número total de alelos representado pela variável $N_{int}$ e não do número de alelos dos genes ativos. Portanto, a mutação se repete ʎ vezes, e todo o processo da estratégia é executado e enviado à decodificação até que um dos critérios de interrupção seja atingido.

Etapa (5) – finalização: na sua fase final é gerado um arquivo (arquivo.mat) com todas as etapas do processo CGP; o *grid* com os genes ativos do melhor cromossomo é mostrado na tela do computador, juntamente com o gráfico (gerações × média dos erros) e as imagens $Im_1$, $Im_2$, $Im_{alvo}$ e a encontrada pela melhor sequência de operadores (melhor cromossomo). É importante ressaltar que o SHAIP-CGP executa seu treinamento com uma amostra de cada vez, ou seja, assim que o sistema treinamento é finalizado (etapa 5 da Figura 4.8), o usuário deve fornecer iniciar todo o procedimento substituindo as imagens e também os parâmetros que desejar.

# Capítulo 5

# *RESULTADOS*

Nesta seção são descritos alguns estudos de caso, a partir dos quais o simulador foi testado para resolução de problemas referentes a processamento de imagens, e está organizada com a seção 5.1, denominada Estudos de Casos, e com seis subseções: subseção 5.1.1, que descreve os conjuntos de treinamento; subseção 5.1.2, onde são apresentadas as entradas do sistema para o treinamento; subseção 5.1.3, que exemplifica as variáveis (parâmetros) do sistema; subseção 5.1.4, na qual são destacadas as estratégias de treinamento do sistema; subseção 5.1.5, que ilustra um exemplo de solução do sistema; subseção 5.1.6, onde se apresenta uma análise comparativa entre os resultados dos experimentos que utilizam o sistema deste trabalho e um sistema baseado em GP linear (IFbyGP), desenvolvido por Pedrino (PEDRINO et al,. 2013).

## 5.1 Estudos de Caso

Todos os estudos de caso foram executados em um computador pessoal com a seguinte configuração de *hardware*: Intel i7 / 2,8 GHz e 12GB de RAM. Tanto o sistema apresentado neste trabalho, quanto o sistema utilizado para análise comparativa (*IFbyGP*) foram implementados em linguagem de programação *MATLAB*. Esta subseção descreve três estudos de caso com foco em processamento de imagens binárias, que foram realizados com o objetivo de avaliar o desempenho do simulador de arquitetura proposto neste trabalho de mestrado. A principal motivação para a concepção dos três estudos de caso (envolvendo diferentes resoluções) foi investigar a eficiência do sistema em detectar automaticamente o padrão de imagens binárias representadas por diferentes objetos em diferentes resoluções.

### 5.1.1 Variáveis Utilizadas nos Estudos de Caso

A introdução de objetos, com formas diferentes e resoluções diferentes, como imagens de treinamento foi feita com a intenção de explorar o reconhecimento de padrões. Ao aumentar o valor do parâmetro λ (número de descendentes apresentado na Tabela 5.1), o sistema vai lidar com um número maior de indivíduos (cromossomo) a cada nova geração e, como consequência, a possibilidade de identificar individualmente a identidade do mais apto a ser passada para a próxima geração vai aumentar.

Os operadores lógicos e morfológicos estão descritos na Tabela 4.1 (subseção 4.1.3), e a função de custo é dada pela equação 4.1 (subseção 4.1.7). Como citado anteriormente, as variáveis de inicialização do sistema são representadas por números inteiros positivos; assim, na Tabela 5.1, são apresentadas as variáveis e seus respectivos valores para suprir todos os testes feitos nos diferentes estudos de caso.

**Tabela 5.1 – Variáveis utilizadas nos treinamentos do SHAIP-CGP.**

| **Parâmetros CGP** | **Variáveis** | **Valores (*default*)** |
|---|---|---|
| Nº de descendentes (filhos) | $\lambda$ | 4;8;16 |
| Nº de cromossomos para iniciar o sistema | $N_{crom}$ | 1 |
| Nº de linhas | $N_L$ | 2;4;8 |
| Nº de colunas | $N_C$ | 4;4;8 |
| Nº de genes | $N_{genes}$ | 8;16;32 |
| Nº de inteiros | $N_{int}$ | 24;48;96 |
| Nº de imagens de entrada | $N_{im}$ | 2 |
| Nº de imagens alvo | $N_{alvo}$ | 1 |
| Nº de elementos estruturantes | $N_{EE}$ | 512 |
| Nº de entradas do sistema | $N_E$ | 514 |
| Nº de saídas do sistema | $N_S$ | 1 |
| Nº de erros | $N_{erro}$ | 0 |
| Nº máximo de gerações | $N_{mg}$ | 2000 |
| Taxa de mutação (%) | $M_{taxa}$ | 10% |
| Nº de operadores | $N_{Op}$ | 8 |

Na Tabela 5.2, alguns parâmetros estão destacados com a cor azul, ou seja, são as principais variáveis do sistema que foram modificadas a cada treinamento do SHAIP-CGP, de acordo com as necessidades impostas pelo usuário, e que

determinam as variações de comportamento durante a evolução. Por exemplo, para mudar o tamanho do cromossomo, é preciso mudar o número de genes que depende do número de linhas e colunas.

### 5.1.2 Conjuntos de Treinamento

A introdução de objetos com formas diferentes nas imagens foi feita com a intenção de explorar as mudanças no tamanho dos cromossomos resultantes, ou seja, o número de operadores necessários e, também, avaliar qual o impacto da utilização de gerações com tamanhos diferentes.

Para essas experiências foram usados três conjuntos de treinamento, $CT_1$, $CT_2$ e $CT_3$, com diferentes tipos de objetos dentro das imagens (quadrados, círculos, estrelas e anéis), como ilustrado na Figura 5.1, e conforme usado em (PEDRINO, et al., 2013)

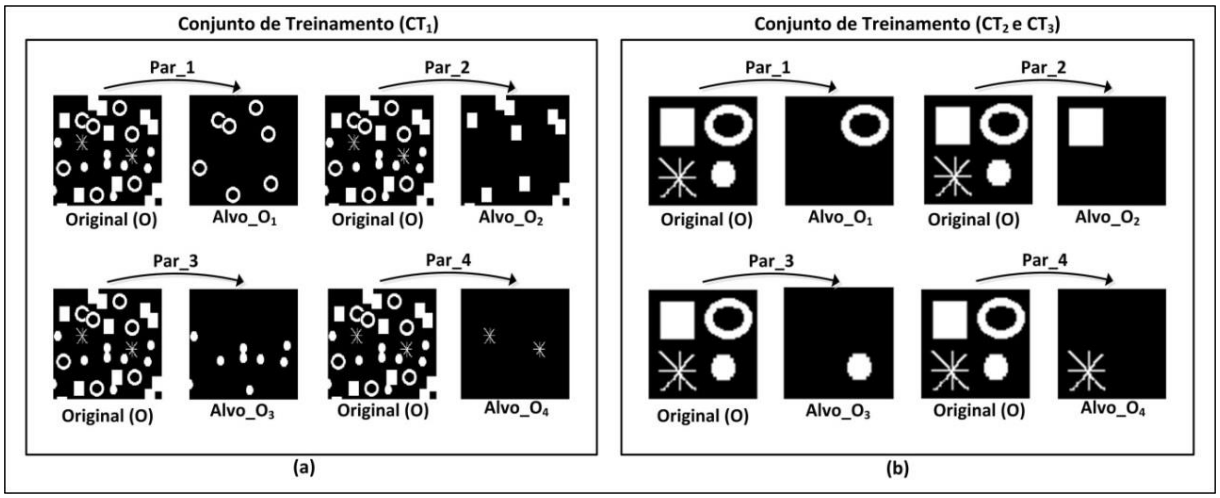



**Figura 5.1 – Conjuntos de treinamento ($CT_1$, $CT_2$, $CT_3$).**

A Figura 5.1 está dividida em duas imagens: (a) representando o conjunto de treinamento ($CT_1$), contendo quatro pares de imagens (O - Alvo_$O_1$), (O - Alvo_$O_2$), (O - Alvo_$O_3$) e (O - Alvo_$O_4$), com resolução de 256×256 *pixels;* (b) representando os conjuntos de treinamento $CT_2$ e $CT_3$, com quatro pares de imagens (O - Alvo_$O_1$), (O - Alvo_$O_2$), (O - Alvo_$O_3$) e (O - Alvo_$O_4$) com resoluções de 32×32 *pixels* e 64×64 *pixels*, respectivamente.

### 5.1.3 Entradas do Sistema para o Treinamento

O treinamento foi feito utilizando-se 12 pares (quatro pares em $CT_1$, quatro pares em $CT_2$, quatro pares em $CT_3$) de imagens (Original-Alvo), conforme mostrado na Figura 5.1, tendo em conta as três possíveis resoluções, ou seja, (32×32, 64×64 e 256×256). Cada par do treinamento é utilizado como entrada do simulador individualmente, ou seja, assim que o simulador encontra a melhor sequência de operadores através de um dos critérios de parada, ele finaliza o processo, e um novo par de imagens deve ser inserido pelo usuário para o próximo treinamento do sistema. A Figura 5.2 ilustra as entradas do SHAIP-CGP, nos experimentos, na qual quatro tipos de objetos com formas diferentes foram utilizados (anel, quadrado, círculo e estrela). Nas resoluções 32×32 64×64, ambas ilustradas na Figura 52.a, apenas uma amostra do objeto é apresentada, diferentemente da resolução 256×256 que são várias amostras do mesmo objeto na mesma imagem.

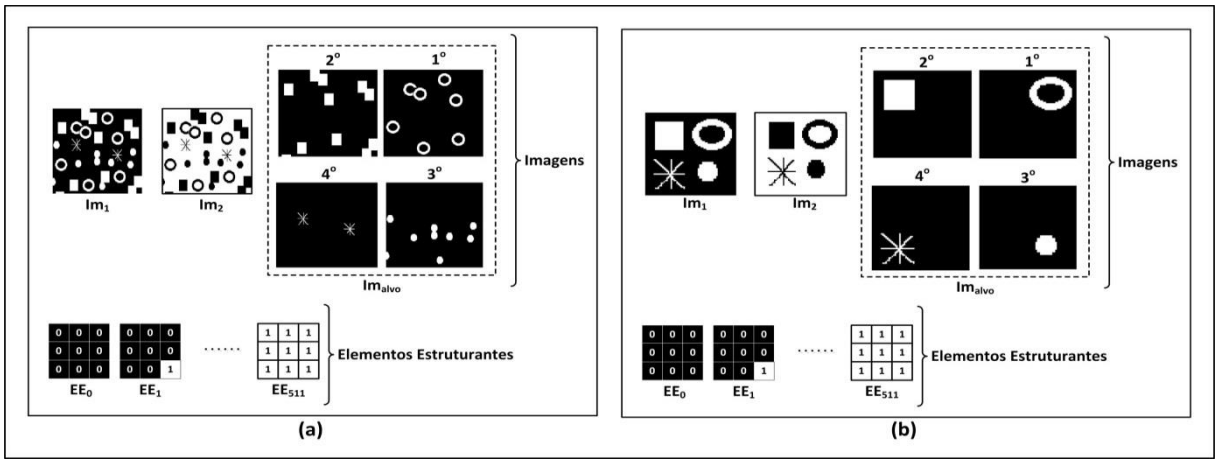



**Figura 5.2 – Entradas do SHAIP-CGP para o treinamento.**

Como citado anteriormente, para o treinamento do sistema foram utilizados três tipos de resoluções diferentes para as imagens. Na Figura 5.2a são mostradas as entradas do sistema (referente ao conjunto de treinamento $CT_1$), que basicamente são as imagens Im1, Im2 (que é Im1 invertida) e as imagens alvo, nomeadas como $Im_{alvo}$ e os elementos estruturantes. Na Figura 5.2b são ilustradas as imagens Im1, Im2 (Im1 invertida), imagens alvo ($Im_{alvo}$) e os elementos estruturantes, retirados dos conjuntos de treinamento $CT_2$, $CT_3$. É possível perceber que, na área tracejada, tanto na Figura 5.2a, quanto na Figura 5.2b, as imagens-alvo estão ordenadas do 1º ao 4º lugar, o que indica que cada imagem é utilizada de uma vez (obedecendo esta ordem) para o

treinamento do sistema. Portanto, todas as imagens são preparadas por um especialista humano.

### 5.1.4 Estratégia de Treinamento do SHAIP-CGP

A estratégia de treinamento foi desenvolvida da seguinte forma: para cada um dos 12 pares de treinamento (três resoluções × quatro pares cada), o sistema foi executado cinco vezes sem qualquer mudança, permitindo que no final a média e o desvio-padrão de todos os casos fossem calculados. A estratégia de treinamento foi repetida considerando três tamanhos diferentes de cromossomos, isto é, $N_{genes}$ = 8 (2 linhas × 4 colunas), $N_{genes}$ = 16 (4 linhas × 4 colunas) e $N_{genes}$ = 32 (4 linhas × 8 colunas), e diferentes valores da variável ʎ (isto é, 4, 8 e 16). Portanto, foram efetuadas 540 evoluções no simulador. É importante ressaltar que a escolha dos valores desses parâmetros está relacionada a outros testes feitos anteriormente (com cromossomos maiores e menores que os apresentados nesses treinamentos descritos), sempre visando facilitar a implementação em *hardware*, que necessita de uma maior simplicidade para não aumentar a complexidade do *hardware* e o custo do projeto. Na Tabela 5.2, são mostrados a média dos erros (%) (M), o desvio-padrão (DP) e a média dos tempos em segundos (MT) dos cinco erros encontrados, para cada par apresentado, tendo em conta: as imagens-alvo (anel, quadrado, círculo e estrela) nas suas diversas resoluções, o número de descendentes ʎ (conforme citado anteriormente) e o tamanho dos cromossomos calculado pelo número de linhas $N_L$ × o número de colunas $N_c$ (c = {8, 16, 32}). Com base nos dados apresentados na Tabela 5.2, pode ser observado que os melhores resultados foram obtidos utilizando-se o cromossomo com tamanho 32 (4 linhas × 8 colunas) e ʎ com valor de 16. Essa não é uma surpresa, considerando-se que o espaço de soluções aumentou e, também, o simulador tem um número maior de cromossomos a cada geração para determinar o indivíduo mais apto (menor erro). É importante lembrar que, quanto maior o valor de ʎ envolvido no processo, maior o tempo de execução (principalmente se o sistema finalizar todo o processo a partir do número de gerações e não com um erro zero). Além disso, com base nos valores do desvio-padrão, pode-se inferir que os resultados têm uma boa precisão, o que garante em sua maioria que as imagens resultantes da aprendizagem do programa evolutivo são muito semelhantes às imagens-alvo. Utilizando-se também os valores apresentados na Tabela 5.2, um gráfico foi

confeccionado para determinar o desempenho do sistema com relação às resoluções (média de erros (c = 8) x média de erros (c = 16) × média de erros (c = 32) × CT's. Esse gráfico é ilustrado pela Figura 5.3.

**Tabela 5.2 – Média, desvio-padrão, dos erros e tempo das evoluções CGP.**

| λ=4 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Estudos de Caso** | | **C = 8 (2 Linhas × 4 Colunas)** | | | **C = 16 (4 Linhas × 4 Colunas)** | | | **C = 32 (4 Linhas × 8 Colunas)** | | |
| | | **(M)** | **(DP)** | **(MT)** | **(M)** | **(DP)** | **(MT)** | **(M)** | **(DP)** | **(MT)** |
| CT1-256x256 | Quadrado | 5,89 | 0,00 | 609,07 | 5,89 | 0,01 | 523,76 | 4,35 | 0,24 | 608,41 |
| | Anel | 5.60 | 0,13 | 477,75 | 5,56 | 0,14 | 510,28 | 1,83 | 1,23 | 623,41 |
| | Estrela | 0,05 | 0,01 | 494,30 | 0,05 | 0,00 | 505,39 | 0,32 | 0,29 | 609,91 |
| | Círculo | 4,10 | 0,02 | 490,06 | 4,11 | 0,03 | 522,19 | 3,90 | 0,26 | 624,90 |
| CT1-64x64 | Quadrado | 5,49 | 0,54 | 274,03 | 5,25 | 0,50 | 304,86 | 3,48 | 0,29 | 444,93 |
| | Anel | 3,61 | 0,12 | 266,75 | 4,41 | 0,65 | 562,89 | 4,54 | 4,21 | 342,36 |
| | Estrela | 0,13 | 0,30 | 124,62 | 0,00 | 0,00 | 48,40 | 1,75 | 1,22 | 345,33 |
| | Círculo | 4,07 | 0,31 | 269,93 | 4,07 | 0,30 | 289,55 | 3,65 | 0,40 | 372,73 |
| CT1-32x32 | Quadrado | 4,59 | 0,27 | 302,94 | 4,63 | 0,28 | 314,65 | 0,25 | 0,57 | 170,93 |
| | Anel | 3,69 | 1,29 | 305,00 | 3,30 | 0,44 | 332,30 | 6,68 | 2,53 | 347,33 |
| | Estrela | 1,05 | 0,14 | 243,44 | 0,29 | 0,27 | 223,43 | 2,70 | 0,52 | 341,23 |
| | Círculo | 3,91 | 0,40 | 269,74 | 3,91 | 0,44 | 279,41 | 3,17 | 2,34 | 308,78 |
| λ=8 | | | | | | | | | | |
| **Estudos de Caso** | | **C = 8 (2 Linhas × 4 Colunas)** | | | **C = 16 (4 Linhas × 4 Colunas)** | | | **C = 32 (4 Linhas × 8 Colunas)** | | |
| | | **(M)** | **(DP)** | **(MT)** | **(M)** | **(DP)** | **(MT)** | **(M)** | **(DP)** | **(MT)** |
| CT1-256x256 | Quadrado | 5,87 | 0,02 | 679,51 | 5,89 | 0,01 | 740,92 | 4,16 | 0,36 | 894,51 |
| | Anel | 5,35 | 0,34 | 672,04 | 5,58 | 0,26 | 748,90 | 1,58 | 0,76 | 914,40 |
| | Estrela | 2,14 | 2,14 | 678,46 | 0,05 | 0,01 | 701,29 | 0,16 | 0,28 | 640,91 |
| | Círculo | 4,12 | 4,69 | 655,66 | 4,12 | 0,02 | 738,31 | 3,90 | 0,26 | 903,13 |
| CT1-64x64 | Quadrado | 5,51 | 0,51 | 290,18 | 5,70 | 0,25 | 357,80 | 3,33 | 0,19 | 452,15 |
| | Anel | 4,56 | 0,76 | 307,99 | 4,02 | 0,89 | 376,51 | 1,62 | 3,49 | 382,74 |
| | Estrela | 2,41 | 0,28 | 164,79 | 0,00 | 0,00 | 27,11 | 0,68 | 1,23 | 332,63 |
| | Círculo | 4,10 | 4,00 | 308,75 | 4,02 | 0,34 | 358,86 | 3,87 | 0,42 | 467,44 |
| CT1-32x32 | Quadrado | 4,59 | 0.37 | 295,17 | 4,37 | 0,58 | 328,63 | 0,00 | 0,00 | 47,49 |
| | Anel | 3,36 | 0,83 | 288,34 | 3,14 | 0,41 | 337,06 | 6,38 | 3,19 | 437,04 |
| | Estrela | 1,95 | 1,38 | 238,47 | 1,33 | 1,50 | 271,56 | 2,34 | 1,31 | 359,32 |
| | Círculo | 4,47 | 1,40 | 285,92 | 3,79 | 0,49 | 335,29 | 4,08 | 2,23 | 410,95 |
| λ=16 | | | | | | | | | | |
| **Estudos de Caso** | | **C = 8 (2 Linhas × 4 Colunas)** | | | **C = 16 (4 Linhas × 4 Colunas)** | | | **C = 32 (4 Linhas × 8 Colunas)** | | |
| | | **(M)** | **(DP)** | **(MT)** | **(M)** | **(DP)** | **(MT)** | **(M)** | **(DP)** | **(MT)** |
| CT1-256x256 | Quadrado | 5,88 | 0,02 | 1054,44 | 5,89 | 0,04 | 1180,19 | 3,93 | 0,38 | 1470,86 |
| | Anel | 5,44 | 0,24 | 1042,22 | 5,14 | 0.26 | 1182,11 | 2,23 | 2,44 | 1502,37 |
| | Estrela | 0,05 | 0,01 | 1015,58 | 0,04 | 0,00 | 1120,68 | 0,00 | 0,00 | 300,64 |
| | Círculo | 4,14 | 0,03 | 1054,78 | 4,11 | 0,03 | 1176,77 | 3,59 | 0,11 | 1522,09 |
| CT1-64x64 | Quadrado | 5,58 | 0,53 | 361,17 | 5,18 | 0,58 | 431,95 | 3,31 | 0,16 | 668,44 |
| | Anel | 4,36 | 0,97 | 354,56 | 3,85 | 1,06 | 452,44 | 0,11 | 0,10 | 363,29 |
| | Estrela | 0,00 | 0,00 | 29,05 | 0,57 | 1,27 | 110,23 | 0,98 | 1,16 | 546,40 |
| | Círculo | 4,14 | 0,28 | 362,88 | 3,89 | 0,26 | 436,47 | 3,48 | 0,27 | 645,38 |
| CT1-32x32 | Quadrado | 4,38 | 0,67 | 323,08 | 4,41 | 0,59 | 403,74 | 0,00 | 0,00 | 115,23 |
| | Anel | 2,81 | 0,73 | 317,74 | 3,28 | 1,30 | 403,08 | 3,48 | 3,98 | 455,25 |
| | Estrela | 0,70 | 0,38 | 295,43 | 0,02 | 0,04 | 179,44 | 2,34 | 1.31 | 574,94 |
| | Círculo | 4,24 | 0,57 | 333,62 | 3,54 | 0,39 | 373,88 | 4,10 | 2,18 | 574,94 |

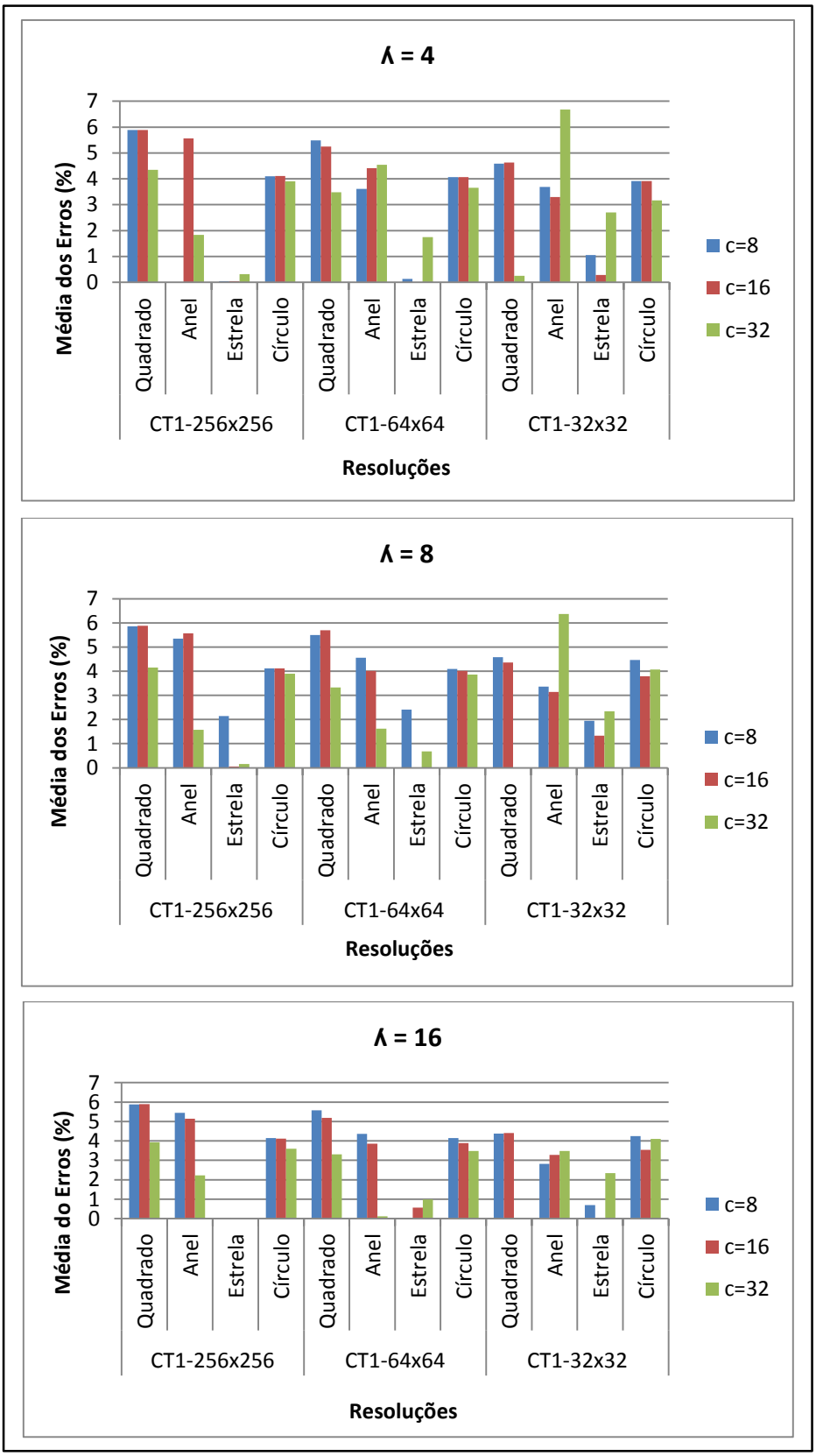



**Figura 5.3 – Gráficos do desempenho do SHAIP-CGP.**

Na Figura 5.3 são mostrados três gráficos de colunas (λ =4, λ =8 e λ =16) com o desempenho do SHAIP-CGP para detectar os quatro padrões (objetos). Pelas

informações contidas neste gráfico, percebe-se que o simulador teve um comportamento com maior êxito reconhecendo o objeto-estrela em todas as resoluções e tamanhos de cromossomo. Considerando-se que a média dos erros está entre zero e sete por cento e que as imagens que possuem objetos com resoluções menores são mais passíveis de serem identificados pelo SHAIP-CGP como outros tipos de objetos (por exemplo, dentro do anel é possível identificar também um círculo) é possível perceber que a partir dos operadores utilizados pelo SHAIP-CGP, o algoritmo mostrou ter uma tendência em adaptar-se ao reconhecimento de imagens com objetos de resoluções maiores.

### 5.1.5 Exemplo de uma Solução

Este exemplo (ilustração gráfica) é o resultado do processo de treinamento para o reconhecimento do objeto-estrela (resolução de 64×64 *pixels*). Na Figura 5.4, são demonstradas: entradas do sistema ($Im_1$, $Im_2$, $Im_{alvo,}$ $EE_0$ – $EE_{511}$); gráfico de evolução (custo: erros × gerações); rede (*grid* 4x8); formação do grid com destaque para os genes ativos que formam o cromossomo; cromossomo na forma de *string* destacando os operadores utilizados; saída final (imagem encontrada).

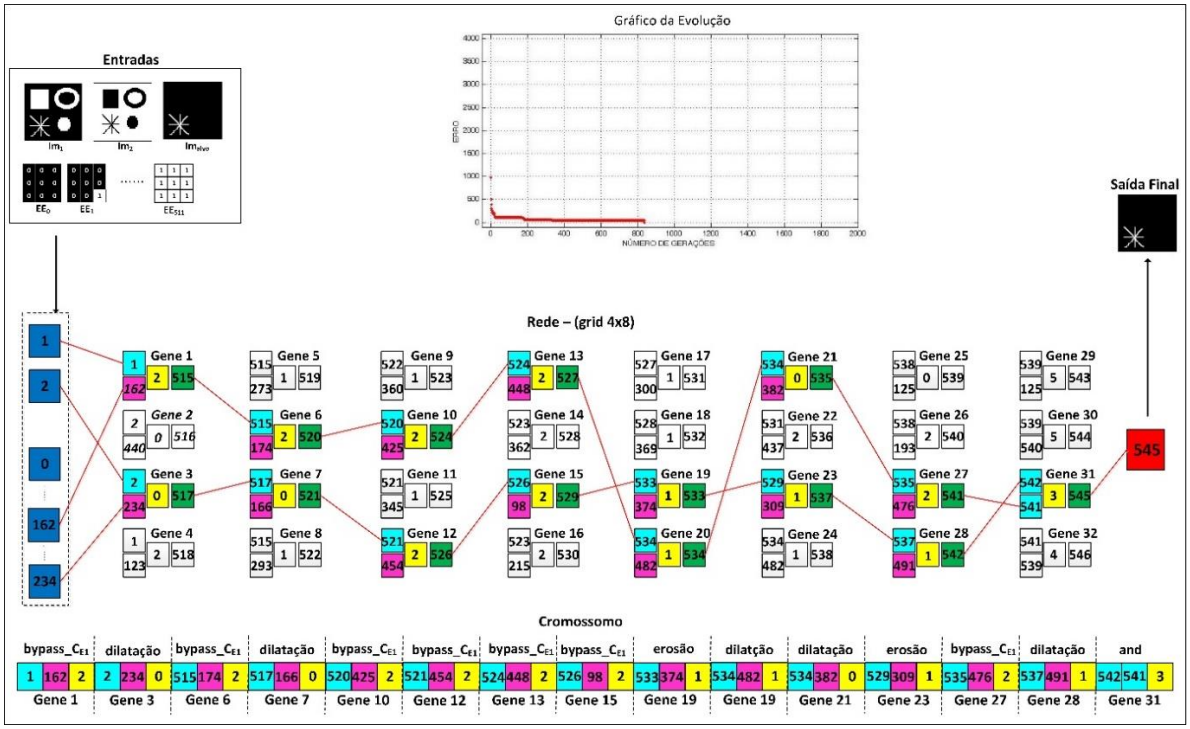



**Figura 5.4 – Exemplo de uma evolução do SHAIP-CGP.**

Para esta evolução do sistema em específico, foram utilizados os parâmetros da Tabela 5.1, sendo que os principais são ($N_L$ = 4, $N_C$ = 8, Ngenes = 32, $N_{crom}$ = 1, ʎ = 16, $N_{mg}$ = 2000, $M_{taxa}$ = 0,1). Como pode ser observado na Figura 5.4, o objeto-alvo foi alcançado, por meio da aplicação da melhor sequência de operadores, ou seja, esta é a combinação perfeita para encontrar a imagem-alvo. Os valores dos erros foram decrescendo a partir do primeiro erro encontrado (próximo de “2000”) a ”0”, após 837 gerações, aproximadamente. Também é interessante dizer que a saída final é dada pela conexão de saída do gene número 31, que recebeu em suas entradas ($CE_1$ e $CE_2$) dos genes 27 e 28, respectivamente. Após ambos os caminhos serem traçados de trás para frente, o sistema destaca o cromossomo inferido pelo procedimento CGP e que é composto por 15 genes. As linhas que conectam os pares de genes entre as colunas mostram os genes ativos no *grid*, e aqueles que tiveram os melhores resultados, ou seja, aqueles que produziram a imagem de saída considerada a mais próxima possível da imagem-alvo.

### 5.1.6 Comparando os Resultados: IFbyGP × SHAIP-CGP

Como mencionado anteriormente neste trabalho, os resultados obtidos pelo sistema evolutivo (IFbyGP), desenvolvido por Pedrino (PEDRINO et al., 2013) (veja seção 3.7), e que é baseado em programação genética, foram utilizados para fins de comparação com os resultados obtidos pelo sistema proposto neste trabalho de mestrado. Embora ambos os sistemas tenham a mesma finalidade, eles possuem diferenças intrínsecas, o que torna difícil estabelecer uma comparação justa. No entanto, como os resultados de ambos estão facilmente disponíveis, foi possível fazer uma comparação mais aprofundada. Assim, esses resultados que utilizam a mesma função de custo e realizam a mesma tarefa, são apresentados na Tabela 5.3. Os resultados mostram diferentes tamanhos de cromossomos utilizados por IFbyGP e pelo SHAIP-CGP, conforme mostrados na comparação. Valores diferentes para ʎ (ou seja, 4, 8 e 16), porém, só foram utilizados pelo SHAIP-CGP, diferentemente do IFbyGP que não usa o lambda em sua estratégia evolucionária. A análise foi realizada com as seguintes regras: (a) IFbyGP com cromossomo c=8 contra o SHAIP-CGP com cromossomo c=8, mas com os valores de ʎ fixados em 4,8,16; (b) IFbyGP com cromossomo c = 16 contra o SHAIP-CGP com cromossomo c = 16, mas com os valores de ʎ fixados em 4,8,16; (c) IFbyGP com cromossomo c = 32 contra SHAIP-

CGP com cromossomo c = 32, mas com os valores de ʎ fixados em 4,8,16. É importante ressaltar que todas as imagens possuem seus comprimentos e larguras iguais; assim, o número total de pixels nas imagens é determinado pela multiplicação (comprimento × largura). Diante disso, as imagens com resolução de 256×256 possuem no total 65536 *pixels;* as imagens com resolução de 64×64 possuem no total 4096 *pixels;* e as imagens com resolução de 32×32 possuem no total 1024 *pixels*, assim todos os valores apresentados na Tabela 5.3 são as médias em porcentagem do erros.

**Tabela 5.3 – IFbyGP × SHAIP-CGP, onde c: tamanho do cromossomo e ʎ: filhos.**

| Estudos de Caso | | Regra (a) | | | | Regra (b) | | | | Regra (c) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | IFbyGP | SHAIP - CGP | | | IFbyGP | SHAIP - CGP | | | IFbyGP | SHAIP - CGP | | |
| | | c=8 | c=8 ; ʎ=4 | c=8 ; ʎ=8 | c=8 ; ʎ=16 | c=16 | c=16 ; ʎ=4 | c=16 ; ʎ=8 | c=16 ; ʎ=16 | c=32 | c=32 ; ʎ=4 | c=32 ; ʎ=8 | c=32 ; ʎ=16 |
| 256×256 | Quadrado | 3,53 | 5,89 | 5,84 | 5,84 | 1,48 | 5,89 | 5,87 | 5,84 | 1,81 | 4,01 | 3,78 | 3,56 |
| | Anel | 5,86 | 5,43 | 4,84 | 5,32 | 5,03 | 5,36 | 5,14 | 4,88 | 5,71 | 0,96 | 1,00 | 0,72 |
| | Estrela | 0,05 | 0,04 | 0,04 | 0,04 | 0,10 | 0,05 | 0,04 | 0,04 | 0,05 | 0,09 | 0,00 | 0,00 |
| | Círculo | 4,00 | 4,07 | 4,08 | 4,11 | 3,20 | 4,07 | 4,11 | 4,08 | 2,83 | 3,56 | 3,61 | 3,49 |
| **Média** | | **3,36** | **3,86** | **3,70** | **3,83** | **2,45** | **3,84** | **3,79** | **3,71** | **2,60** | **2,16** | **2,10** | **1,94** |
| **Desvio-Padrão** | | **2,43** | **2,66** | **2,54** | **2,63** | **2,14** | **2,64** | **2,60** | **2,55** | **2,37** | **1,92** | **1,89** | **1,85** |
| 64×64 | Quadrado | 3,86 | 4,64 | 4,71 | 4,64 | 3,83 | 4,64 | 5,25 | 4,69 | 1,97 | 3,30 | 3,22 | 3,22 |
| | Anel | 3,08 | 3,49 | 3,47 | 3,54 | 2,8 | 3,64 | 3,42 | 3,34 | 2,25 | 0,00 | 0,00 | 0,00 |
| | Estrela | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,30 | 3,20 | 0,00 | 0,00 |
| | Círculo | 4,00 | 3,69 | 3,91 | 3,64 | 3,4 | 3,64 | 3,64 | 3,61 | 2,89 | 0,00 | 3,42 | 3,25 |
| **Média** | | **2,74** | **2,96** | **3,02** | **2,96** | **2,51** | **2,98** | **3,08** | **2,91** | **1,85** | **1,63** | **1,66** | **1,62** |
| **Desvio-Padrão** | | **1,87** | **2,03** | **2,03** | **1,72** | **1,72** | **2,04** | **2,21** | **2,03** | **1,10** | **1,88** | **1,92** | **1,87** |
| 32×32 | Quadrado | 3,66 | 4,20 | 2,15 | 3,91 | 0,54 | 4,20 | 3,91 | 3,91 | 0,54 | 0,00 | 0,00 | 0,00 |
| | Anel | 3,66 | 2,64 | 4,20 | 2,05 | 2,04 | 2,64 | 2,44 | 2,34 | 4,30 | 2,15 | 0,68 | 0,00 |
| | Estrela | 0,86 | 0,98 | 0,00 | 0,29 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| | Círculo | 1,83 | 3,91 | 3,32 | 3,52 | 0,65 | 3,22 | 3,22 | 3,13 | 0,86 | 1,76 | 0,10 | 0,20 |
| **Média** | | **2,50** | **2,93** | **2,42** | **2,44** | **0,81** | **2,52** | **2,39** | **2,35** | **1,43** | **0,98** | **0,20** | **0,05** |
| **Desvio-Padrão** | | **1,39** | **1,47** | **1,82** | **1,64** | **0,87** | **1,80** | **1,70** | **1,69** | **1,95** | **1,14** | **0,33** | **0,10** |

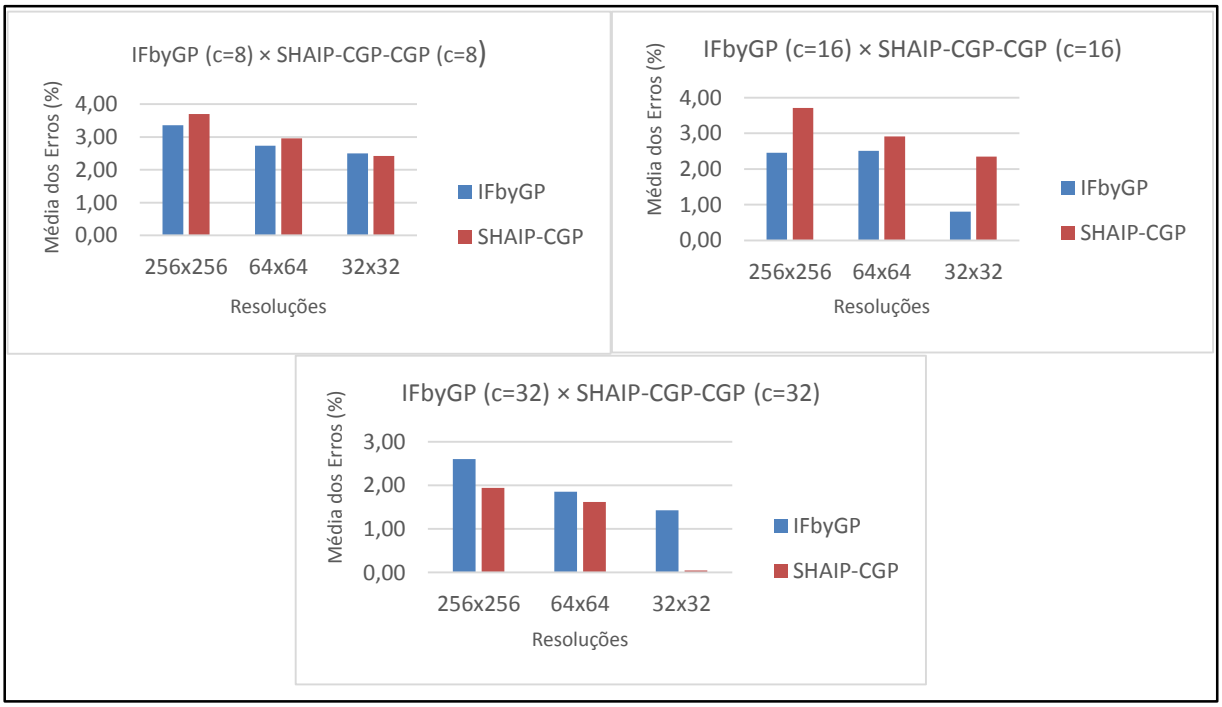



**Figura 5.5 – Gráficos de comparação dos resultados (dados tabela 5.3).**

Analisando os dados da Tabela 5.3 e os seus gráficos correspondentes, torna-se evidente que os cromossomos com tamanho c=32 têm, em quase todos os casos, os melhores valores para o erro médio. Os números relativos ao desvio-padrão apresentaram comportamento semelhante, com os cromossomos c=32 que são os únicos com resultados mais satisfatórios alcançados (considerando esta comparação), o que não quer dizer que os resultados apresentados pelo sistema IFbyGP não sejam considerados satisfatórios pois ele foi desenvolvido para fazer outras tarefas além do reconhecimento de imagens binárias (seção 3.7). O cromossomo com tamanho c = 8 e 16 do SHAIP-CGP tiveram desempenhos piores do que os do IFbyGP em todas as comparações. Uma das razões para o fraco desempenho destes tamanhos é talvez, devido às poucas opções de operadores disponíveis no *grid* para a construção de um filtro satisfatório para todos os objetos. No entanto, quando o simulador de arquitetura utilizou o cromossomo c=32, alcançou o melhor desempenho em todos os estudos de caso. Portanto, se os valores de desvio-padrão são relativamente baixos, é possível concluir então que a dispersão é muito pequena quando comparada com a média dos erros, uma indicação da versatilidade do sistema para se adequar a diferentes tipos de imagens e ainda manter um desempenho aceitável quando se está construindo uma sequência de operadores de imagem.

### 5.1.7 Testando a Sequência de Operadores: aumentando as resoluções

A partir dos resultados encontrados nos estudos de casos citados anteriormente, foi possível realizar três etapas de avaliações do comportamento do sistema com a utilização dos mesmos conjuntos de treinamento, simplesmente aumentando as resoluções em 50%, ou seja, o conjunto de treinamento $CT_1$ deixa de utilizar a resolução de 256×256 e passa a utilizar a resolução de 384×384, o conjunto de treinamento $CT_2$ deixa de utilizar a resolução de 64×64 e passa a utilizar a resolução de 96×96, o conjunto de treinamento $CT_3$, deixa de utilizar a resolução de 32×32, e passa a utilizar a resolução de 48×48. A metodologia utilizada para esta avaliação se resume na escolha de alguns dos melhores resultados (melhor sequência de operadores) e em sua utilização para encontrar uma nova imagem de saída através da ação dos operadores desse cromossomo, em um conjunto de treinamento com maior resolução. Resumindo, deve-se avaliar se a sequência de

operadores que determinou um erro qualquer (o mais próximo de zero) é capaz de encontrar um erro o mais próximo possível do determinado anteriormente, tendo como base a mesma imagem-objeto, mas com resolução diferente. Para melhor entendimento, são apresentadas na sequência desta seção as Tabelas (5.4, 5.5 e 5.6) contendo os dados necessários para a confecção das três Figuras (5.6, 5.7 e 5.8), na qual, todas estão divididas em imagens descritas da seguinte forma: $C_i$, que é o cromossomo devidamente codificado; $Im\_ac_i$, que representa a imagem-alvo originalmente usada para os testes; $Im\_bc_i$, que identifica a imagem encontrada nos primeiros testes (saída final do *grid*); $Im\_cc_i$, que é a imagem encontrada nos testes com a resolução aumentada. O índice "i" presente nos nomes das imagens e no cromossomo representa os números inteiros, que podem variar de um a quatro dependendo da figura.

### 5.1.7.1 Primeira Etapa de Avaliação

Na primeira etapa de avaliação, foram apresentados os resultados encontrados utilizando cromossomos com tamanhos (c=8, c=16, c=32), com $\lambda$=4. Na Figura 5.6, foram apresentados os resultados dos testes efetuados com três dos melhores cromossomos e alguns pares de treinamento com as seguintes formas (objeto-anel, objeto-quadrado, objeto-estrela e objeto-círculo).

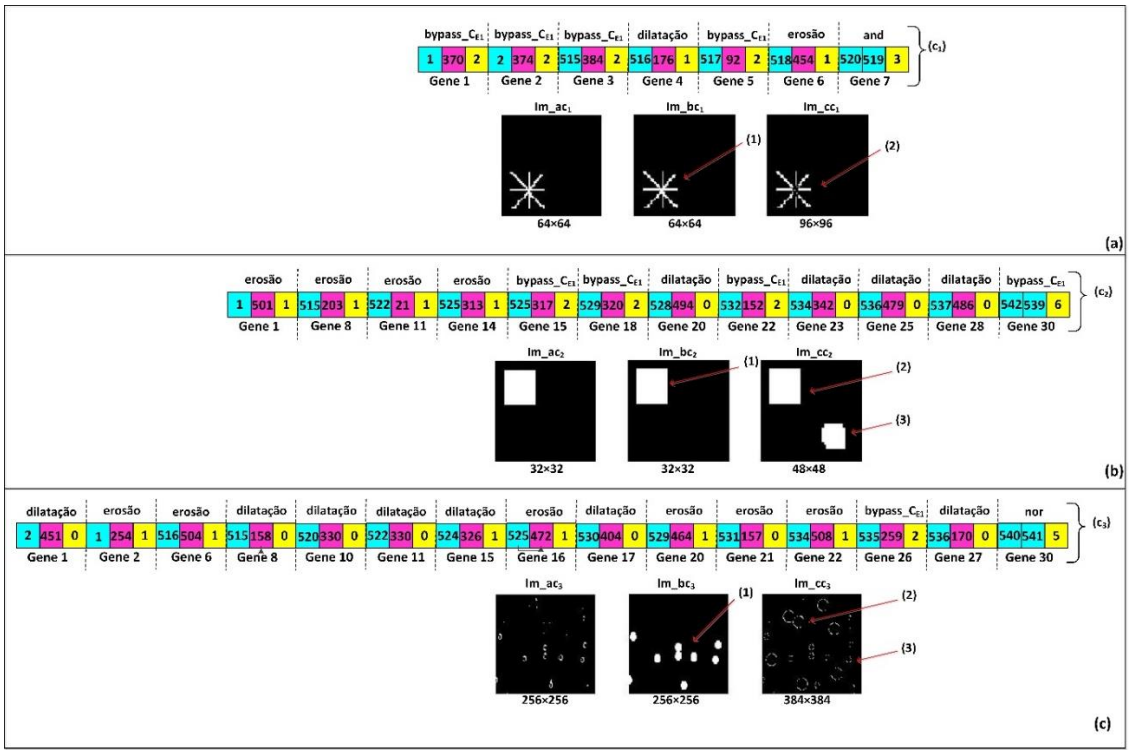



**Figura 5.6 – Ilustração dos resultados da primeira avaliação.**

**Tabela 5.4 – Dados utilizados para obter as imagens da Figura 5.6.**

| Treinamento Inicial | | | | | Teste com Resolução Aumentada em 50% | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Objeto | Tamanho do Cromossomo | Resolução | Erro (pixels) | Erro (%) | Objeto | Tamanho do Cromossomo | Resolução | Erro (pixels) | Erro (%) |
| Estrela (Im_bc1) | c=8 (2 linhas x 4 colunas) | 64×64 | 0 | 0,00 | Estrela (Im_cc1) | c=8 (2 linhas x 4 colunas) | 96×96 | 44 | 0,47 |
| Quadrado (Im_bc2) | c=32 (4 linhas x 8 colunas) | 32×32 | 0 | 0,00 | Quadrado (Im_cc2) | c=32 (4 linhas x 8 colunas) | 48×48 | 123 | 5,38 |
| Círculo (Im_bc3) | c=32 (4 linhas x 8 colunas) | 256×256 | 2335 | 3,56 | Círculo (Im_cc3) | c=32 (4 linhas x 8 colunas) | 384×384 | 7223 | 4,89 |

Os experimentos ilustrados na Figura 5.6 apresentaram as seguintes características específicas. Na Figura 5.6a, foi utilizado um cromossomo com sete genes, que resultou em "Im_bc$_1$", e determinou um erro de 0 *pixels (0%)* em relação à imagem-alvo (Im_ac$_1$ - 64×64). Este mesmo cromossomo, agindo sobre uma imagem de 96×96, encontrou como resultado "Im_cc$_1$", com um erro de 44 *pixels* (0,47%). Na comparação entre (Im_bc$_1$ e Im_cc$_1$) em Im_bc$_1$ uma imagem (1) foi determinada por uma sequência perfeita de operadores para o objeto-estrela, e em Im_cc$_1$, com a resolução maior, encontrou-se uma imagem (2), com um erro muito bom se comparado ao número total de *pixels* da imagem (9216), mostrando que essa sequência de operadores pode ser utilizada para resoluções maiores.

Para a Figura 5.6b, foi utilizado um cromossomo com doze genes, que apresentou como resultado "Im_bc$_2$", e que determinou um erro de 0 *pixels (0%)*, em relação à imagem-alvo (Im_ac$_2$ - 32×32). Este mesmo cromossomo, agindo sobre uma imagem de 48×48, encontrou um resultado mostrado em "Im_cc$_2$" com um erro de 123 *pixels* (5,38%). Na comparação entre os resultados (Im_bc$_2$ e Im_cc$_2$), em Im_bc$_2$ uma imagem (1), foi determinada por uma sequência perfeita de operadores para o objeto quadrado; em Im_cc$_2$, encontrou-se uma imagem idêntica a imagem-alvo (2) e também uma imagem (3) na mesma localização do objeto-círculo, mas com uma forma quadrada.

Finalmente, na Figura 5.6c, foi utilizado um cromossomo com quinze genes, que apresentou como resultado "Im_bc$_3$", e que determinou um erro de 2335 *pixels (3,56%)*, em relação à imagem-alvo (Im_bc$_3$ - 256×256). Este mesmo cromossomo agindo sobre uma imagem de 384×384, apresentou como resultado "Im_cc$_3$", com um erro de 7223 *pixels* (4,89%). Na comparação entre os resultados (Im_bc$_3$ e Im_cc$_3$) em Im_bc$_3$, foram encontradas as bordas do objeto desejado (1), neste caso o círculo; e em Im_cc$_3$ também foram encontradas as mesmas bordas (2) com contornos mais suaves, e a borda do outro objeto anel com forma parecida (3). Portanto, nesta primeira avaliação, o comportamento do SHAIP-CGP, apesar de ter encontrado erros aceitáveis, teve muitas variações, tendo alguma precisão em apenas um dos testes

(Figura 5.6a), o que determina que, com o aumento na resolução das imagens, os operadores tendem a encontrar formas com aparências muito próximas.

### 5.1.7.2 Segunda Etapa de Avaliação

Na segunda etapa, foi utilizada a mesma forma de testes da primeira, sendo que os cromossomos utilizados foram os encontrados nas evoluções com os seguintes parâmetros (c=8, c=16, c=32 e λ = 8) e os objetos (anel, quadrado, estrela). Na Figura 5.7, são ilustradas as imagens encontradas confeccionadas a partir dos dados da Tabela 5.5.

**Tabela 5.5 – Dados utilizados para obter as imagens da Figura 5.7.**

| λ=8 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Treinamento Inicial | | | | | Teste com Resolução Aumentada em 50% | | | | |
| Objeto | Tamanho do Cromossomo | Resolução | Erro (pixels) | Erro (%) | Objeto | Tamanho do Cromossomo | Resolução | Erro (pixels) | Erro (%) |
| Estrela (Im_bc1) | c=16 (4 linhas x 4 colunas) | 64×64 | 0 | 0,00 | Estrela (Im_cc1) | c=16 (4 linhas x 4 colunas) | 96×96 | 35 | 0,37 |
| Quadrado (Im_bc2) | c=32 (4 linhas x 8 colunas) | 32×32 | 0 | 0,00 | Quadrado (Im_cc2) | c=32 (4 linhas x 8 colunas) | 48×48 | 130 | 5,64 |
| Anel (Im_bc3) | c=32 (4 linhas x 8 colunas) | 64×64 | 0 | 0,00 | Anel (Im_cc3) | c=32 (4 linhas x 8 colunas) | 96×96 | 732 | 7.9 |

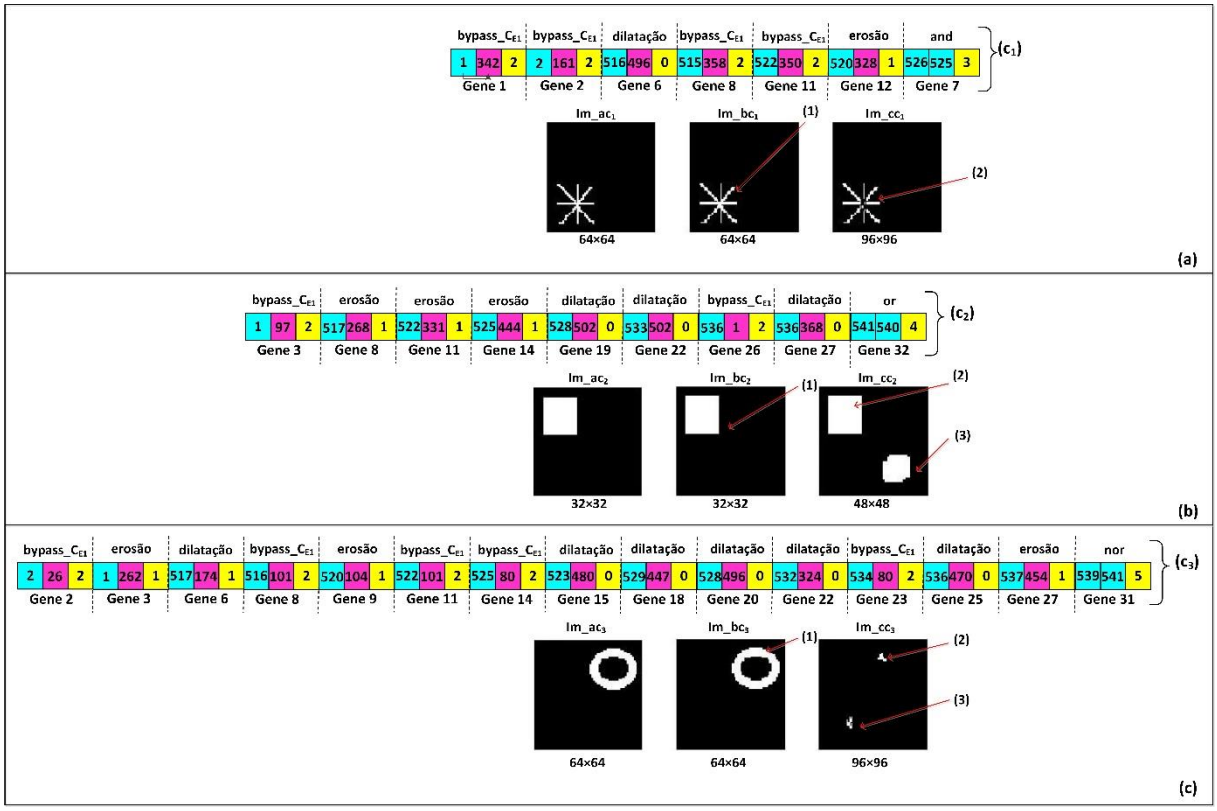



**Figura 5.7 – Ilustração dos resultados da segunda avaliação.**

Os resultados das avaliações, apresentados na Figura 5.7, possuem as seguintes características específicas. Na Figura 5.7a, foi utilizado um cromossomo com sete genes, que resultou em "Im_bc$_1$", e determinou um erro de 0 *pixels (0%)* em

relação à imagem-alvo (Im_ac$_1$ - 64×64). Este mesmo cromossomo, agindo sobre uma imagem de 96×96, encontrou como resultado “Im_cc$_1$”, com um erro de 35 *pixels* (0,37%). Na comparação entre (Im_bc$_1$ e Im_cc$_1$) em Im_bc$_1$ uma imagem (1) foi determinada por uma sequência perfeita de operadores para o objeto-estrela, e em Im_cc$_1$, com maior resolução, encontrou-se uma imagem (2), com um erro aceitável se comparado ao número total de *pixels* da imagem, mostrando que essa sequência de operadores pode ser utilizada para resoluções maiores.

Para a Figura 5.6b, foi utilizado um cromossomo com nove genes, que apresentou como resultado “Im_bc$_2$”, e que determinou um erro de 0 *pixels (0%)*, em relação à imagem-alvo (Im_ac$_2$ - 32×32). Este mesmo cromossomo, agindo sobre uma imagem de 48×48, encontrou um resultado mostrado em “Im_cc$_2$” com um erro de 130 *pixels* (5,64%). Na comparação entre os resultados (Im_bc$_2$ e Im_cc$_2$), em Im_bc$_2$, uma imagem (1) foi determinada por uma sequência perfeita de operadores para o objeto quadrado; em Im_cc$_2$, foram encontradas uma imagem idêntica a imagem-alvo (2) e também uma imagem (3) pertencente ao objeto-círculo.

Para a Figura 5.7c, foi utilizado um cromossomo com quinze genes, que resultou em “Im_bc$_3$” e determinou um erro de 0 *pixels (0%)*, em relação à imagem-alvo (Im_ac$_3$ - 64×64). Este mesmo cromossomo, agindo sobre uma imagem de 96×96 encontrou como resultado “Im_cc$_3$”, com um erro de 732 *pixels* (7.90%). Na comparação entre os resultados (Im_bc$_3$ e Im_cc$_3$), em Im_bc$_3$, uma imagem (1) foi determinada por uma sequência perfeita de operadores para o objeto-anel, e em Im_cc$_3$, foram marcados apenas alguns *pixels* da posição do objeto-anel (2) em relação ao fundo da imagem; além disso, também foram encontrados alguns *pixels* do objeto-estrela, que não eram esperados.

### 5.1.7.3 Terceira Etapa da Avaliação

Na terceira avaliação, foram utilizados os parâmetros (c=8, c=16, c=32 e $\lambda$ = 16) e os objetos (anel, quadrado e estrela) para encontrar os resultados que são ilustrados na Figura 5.8.

**Tabela 5.6 – Dados utilizados para obter as imagens da Figura 5.8.**

| λ=16 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Treinamento Inicial | | | | | Teste com Resolução Aumentada em 50% | | | | |
| Objeto | Tamanho do Cromossomo | Resolução | Erro (pixels) | Erro (%) | Objeto | Tamanho do Cromossomo | Resolução | Erro (pixels) | Erro (%) |
| Anel (Im_bc1) | c=32 (4 linhas x 8 colunas) | 32×32 | 0 | 0,00 | Anel (Im_cc1) | c=32 (4 linhas x 8 colunas) | 48×48 | 130 | 5,60 |
| Quadrado (Im_bc2) | c=32 (4 linhas x 8 colunas) | 32×32 | 0 | 0,00 | Quadrado (Im_cc2) | c=32 (4 linhas x 8 colunas) | 48×48 | 106 | 4,60 |
| Estrela (Im_bc3) | c=32 (4 linhas x 8 colunas) | 256×256 | 0 | 0,00 | Estrela (Im_cc3) | c=32 (4 linhas x 8 colunas) | 384×384 | 374 | 0,20 |

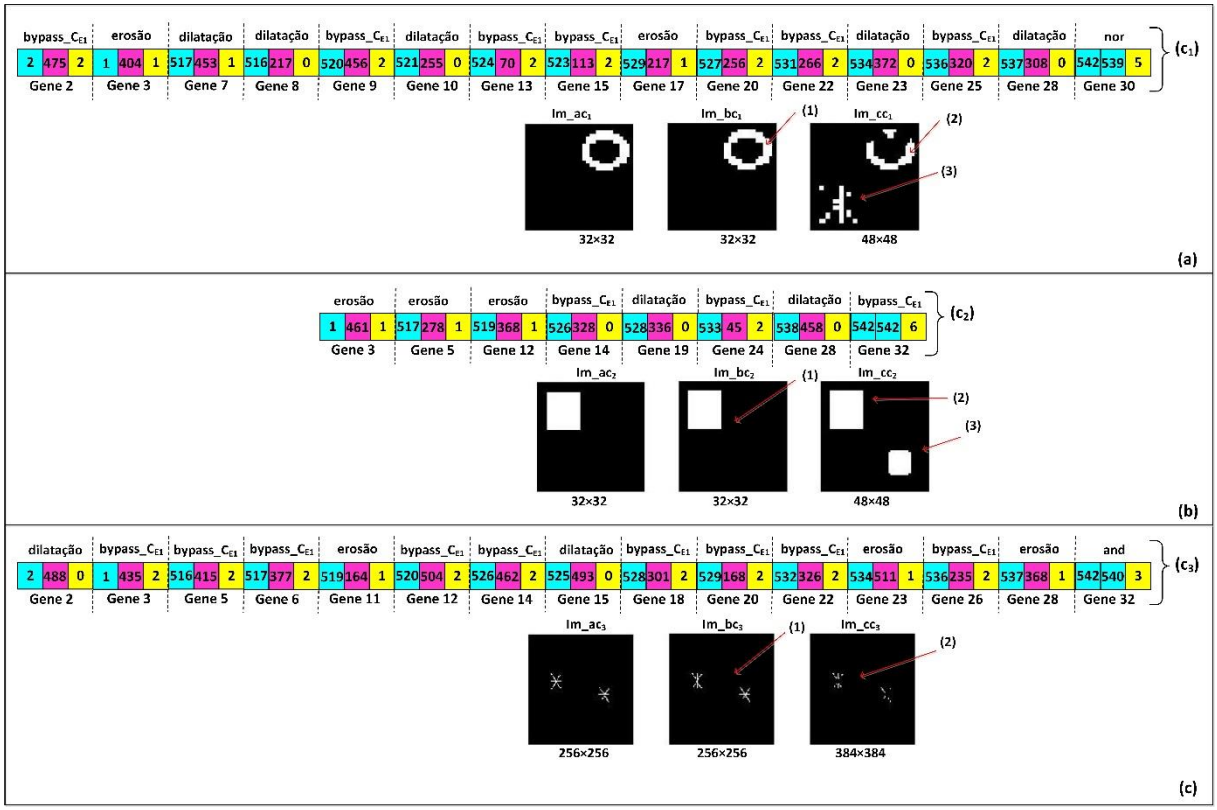



**Figura 5.8 – Ilustração dos resultados da terceira avaliação.**

Os experimentos ilustrados na Figura 5.8 apresentaram as seguintes características. Na Figura 5.8a foi utilizado um cromossomo com quinze genes, que a partir de sua sequência de operadores encontrou como resultado “Im_bc$_1$”, que determinou um erro de 0 *pixels (0,00%)*, em relação à imagem-alvo (Im_ac$_1$ - 32×32); este mesmo cromossomo, agindo sobre uma imagem de resolução 48×48, encontrou o resultado ilustrado em “Im_cc$_1$” com um erro de 130 *pixels* (5,60%), ou seja, se comparado um resultado com o outro (Im_bc$_1$ e Im_cc$_1$) é possível perceber que, em Im_bc$_1$, o SHAIP-CGP encontrou o objeto-anel (1) idêntico a imagem alvo; com o aumento da resolução em Im_cc$_1$, foi encontrado objeto-anel quase em sua totalidade (2) e mais pixels pertencentes ao objeto-estrela (3).

Para a Figura 5.8b foi utilizado um cromossomo com oito genes, que apresentou como resultado “Im_bc$_2$”, e que determinou um erro de 0 *pixels (0%)*, em relação à imagem-alvo (Im_ac$_2$ - 32×32). Este mesmo cromossomo, agindo sobre uma imagem de 48×48, encontrou um resultado mostrado em “Im_cc$_2$” com um erro de 106 *pixels* (4,60%). Na comparação entre os resultados (Im_bc$_2$ e Im_cc$_2$), em Im_bc$_2$, uma imagem (1) foi determinada por uma sequência perfeita de operadores para o objeto

quadrado; em Im_cc$_2$, foram encontradas uma imagem idêntica a imagem-alvo (2) e também uma imagem (3) pertencente ao objeto-quadrado.

Na Figura 5.8c foi utilizado um cromossomo com quinze genes, que resultou em "Im_bc$_3$", e determinou um erro de 0 *pixels (0%)* em relação à imagem-alvo (Im_ac$_3$ - 256×256). Este mesmo cromossomo, agindo sobre uma imagem de 384×384, encontrou como resultado "Im_cc$_3$", com um erro de 374 *pixels* (0,20%). Na comparação entre (Im_bc$_3$ e Im_cc$_3$), em Im_bc$_3$, uma imagem (1) foi determinada por uma sequência perfeita de operadores para o objeto-estrela, e em Im_cc$_3$, determinou mesmo com a resolução maior, uma imagem (2), faltando alguns pixels no centro das duas estrelas e mostrando uma condição aceitável dentro do contexto geral de reconhecimento de padrões

### 5.1.8 Testando Sequência de Operadores: alterações nos objetos

Nesta subseção será apresentado um outo tipo de avaliação do sistema SHAIP-CGP. O procedimento é baseado na utilização de quatro dos melhores cromossomos (melhor sequência de operadores), gerados pelos treinamentos (estudo de casos) descritos anteriormente neste capítulo.

A metodologia utilizada nessa avaliação se resume na utilização dos operadores "escolhidos" (regras CGP) pelo SHAIP-CGP (em cada cromossomo) para fazer o reconhecimento de um objeto (com pequenas alterações na sua forma), pertencente aos mesmos conjuntos de treinamento ($CT_1$, $CT_2$, $CT_3$) utilizados nos estudos de caso, ou seja, avaliar se os mesmos operadores que determinaram uma imagem com "erro zero" são capazes de reconhecer um objeto parecido (na forma e no tamanho) com o outro apresentado na imagem-alvo encontrada anteriormente.

Na Figura 5.9 são apresentados os resultados obtidos nos testes, onde: $C_i$ é o cromossomo devidamente codificado; Im_ac$_i$ é a imagem-alvo original utilizada nos experimentos anteriores para visualizar os objetos antes da modificação; Im_bc$_i$ representa a imagem encontrada sem modificações na sua forma; Im_cc$_i$ é a imagem encontrada nos testes após a ação dos operadores (na imagem original). O índice "i" presente nos nomes das imagens e do cromossomo representa os números inteiros de um a quatro. A Tabela 5.7 contém os dados obtidos a partir destes testes citados anteriormente.

**Tabela 5.7 – Dados utilizados para obter as imagens da Figura 5.9.**

| Treinamento Inicial | | | | | Testes com Alterações nos Objetos | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Objeto | Tamanho do Cromossomo | Resolução | Erro (pixels) | Erro (%) | Objeto | Tamanho do Cromossomo | Resolução | Erro (pixels) | Erro (%) |
| Anel (Im_bc1) | c=32 (4 linhas x 8 colunas) | 64×64 | 0 | 0,00 | Anel (Im_cc1) | c=32 (4 linhas x 8 colunas) | 64×64 | 101 | 2,40 |
| Quadrado (Im_bc2) | c=32 (4 linhas x 8 colunas) | 32×32 | 0 | 0,00 | Quadrado (Im_cc2) | c=32 (4 linhas x 8 colunas) | 32×32 | 7 | 0,60 |
| Círculo (Im_bc3) | c=32 (4 linhas x 8 colunas) | 32×32 | 0 | 0,00 | Círculo (Im_cc3) | c=32 (4 linhas x 8 colunas) | 32×32 | 7 | 0,68 |
| Estrela (Im_bc4) | c=32 (4 linhas x 8 colunas) | 256×256 | 0 | 0,00 | Estrela (Im_cc4) | c=32 (4 linhas x 8 colunas) | 256×256 | 143 | 0,21 |

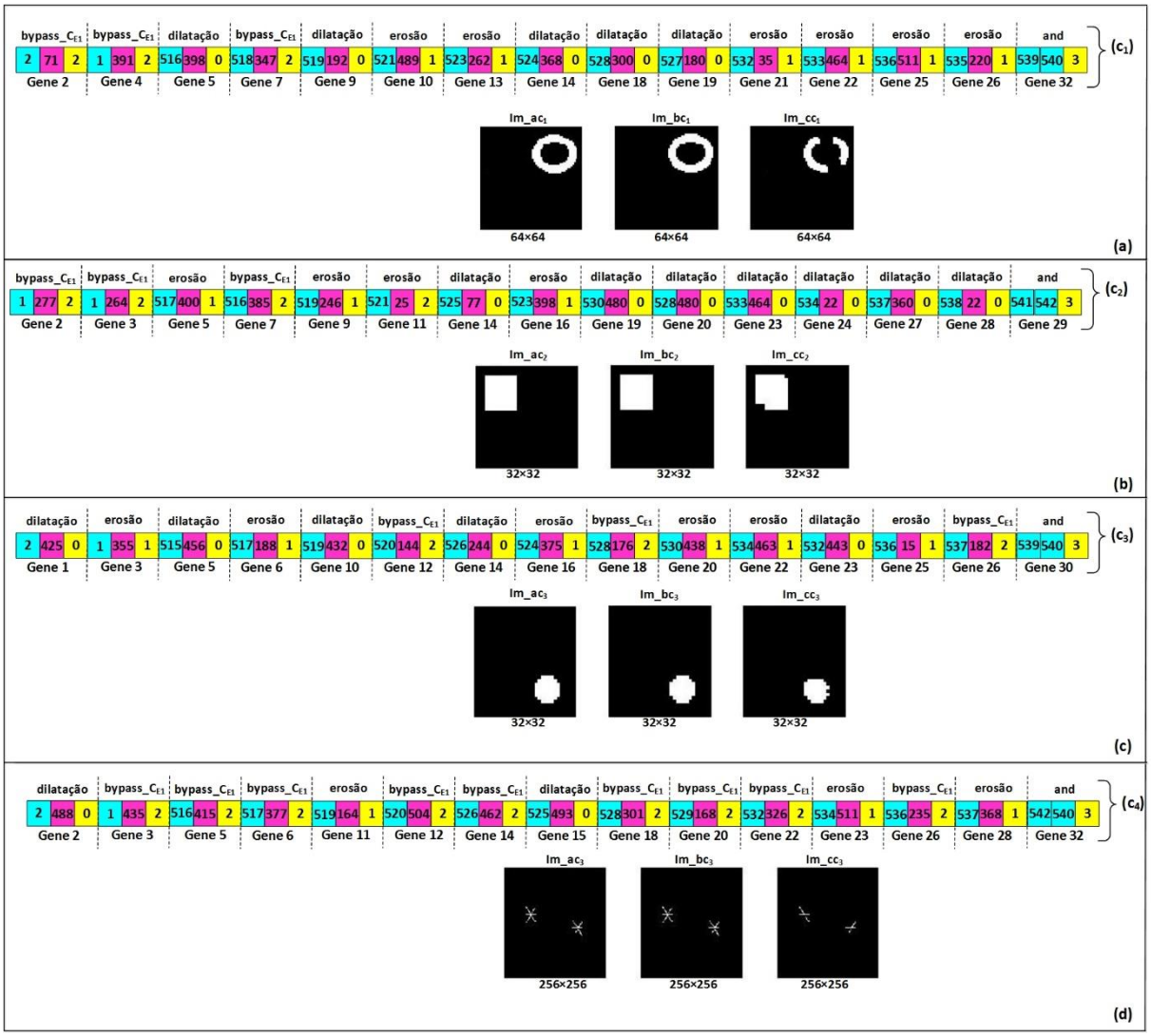



**Figura 5.9 – Ilustração dos resultados da avaliação com alterações nos objetos.**

Os resultados apresentados na Figura 5.9 podem ser descritos da seguinte forma: Na Figura 5.9a foi utilizado para o teste um cromossomo com quinze genes, codificados com operadores de *erosão, dilatação, bypass_C$_{E1}$ e and*, e que encontrou um erro zero para o objeto-anel nos treinamentos anteriores; na imagem Im_cc$_1$, é possível perceber a diferença na forma do objeto-anel (em relação à imagem Im_ac$_1$), anteriormente reconhecida pelo cromossomo C$_1$. Em Im_cc$_1$, é possível perceber que os operadores do cromossomo foram o suficiente para encontrar o objeto-anel com 101 *pixels* (2,40%) de diferença do mesmo da imagem Im_ac$_1$;

Na Figura 5.9b foi utilizado para o teste um cromossomo com quinze genes, codificados por operadores de *erosão, dilatação, bypass_$C_{E1}$ e and*, no qual também se encontrou um erro zero para o objeto-quadrado nos treinamentos anteriores. Na imagem Im_$bc_2$, é possível perceber a diferença na forma do objeto-anel (em relação à imagem Im_$ac_2$), reconhecida pelo cromossomo $C_2$. Em Im_$cc_2$ é possível perceber que os operadores do cromossomo conseguiram encontrar o objeto-quadrado com 7 *pixels* (0,60%);

Na Figura 5.9c foi utilizado no teste um cromossomo com quinze genes, codificados com operadores de *erosão, dilatação, bypass_$C_{E1}$ e and*, classificado com um erro zero para o objeto-círculo; nos treinamentos anteriores. Em Im_$cc_3$, é possível perceber que os operadores do cromossomo conseguiram encontrar o objeto-círculo com 7 *pixel* (0,68%) muito próximo do erro zero comparado com Im_$ac_2$;

Por fim, na Figura 5.9d foi utilizado um cromossomo com quinze genes, codificados com operadores como *erosão, dilatação, bypass_$C_{E1}$ e and*, e que encontrou um erro zero para o objeto anel nos treinamentos anteriores; Em Im_$cc_4$, é possível perceber que as operações do cromossomo geraram uma imagem do objeto-estrela com 143 *pixels* (0,21%) de diferença em relação ao mesmo objeto da imagem Im_$ac_4$.

# Capítulo 6

# *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: 6.1 Considerações Finais; 6.2 Contribuições e Limitações; 6.3 Trabalhos Futuros.*

## 6.1 Considerações Finais

Este trabalho propôs a utilização de uma abordagem em Programação Genética Cartesiana (CGP) para a construção de um simulador de arquitetura em *hardware* para a construção automática de sequências de operadores de imagens, destinados à filtragem de imagens binárias. O algoritmo baseado em CGP foi implementado com o nome de SHAIP-CGP, que também pode ser usado em aplicações relacionadas à segmentação de imagens e de reconhecimento de padrões. Nesta dissertação de mestrado foram descritos os resultados obtidos com o simulador em estudos de caso relacionados com à filtragem de imagens binárias (quatro tipos de objetos), assim como, a comparação destes resultados com outros resultados encontrados no artigo (PEDRINO et al., 2013), que aborda uma implementação baseada em Programação Genética Linear para vários tipos de experimentos, sendo que um deles é específico para reconhecimento de imagens binárias.

## 6.2 Contribuições e Limitações

A motivação deste trabalho foi criar um simulador para evolução de filtros de imagens baseado em um sistema evolutivo (CGP) compacto, visando-se no futuro uma implementação em *hardware* para processamento de imagens em tempo real. O objetivo desta pesquisa foi o de explorar o uso da Programação Genética Cartesiana em tarefas de construção automática de filtros para processamento de imagens binárias. Foi possível inferir durante esse período de estudos e experimentos, que a CGP se mostrou muito apropriada na simulação para futuras implementações em *hardware* do sistema, devido a sua forma cartesiana, que se aproxima de um *layout* de circuito (o “fenótipo” propriamente dito). A implementação do sistema SHAIP-CGP é bem simples e compacta, e proporciona ao usuário, de forma flexível, as condições necessárias para as alterações nos parâmetros de inicialização (aumentando a eficiência no processo de aprendizagem do sistema). Os experimentos foram conduzidos utilizando variações nas regras de CGP aliadas a estratégia evolucionária (1+ʎ), e mostraram que tal estratégia proporciona um melhor desempenho relacionado à aptidão. Nas comparações dos resultados entre o SHAIP-CGP e um dos estudos de caso desenvolvidos no trabalho denominado IFbyGP, para imagens binárias, a média dos erros e o desvio-padrão encontrados pelo SHAIP-CGP foram superiores em muitos casos, e mesmo os que foram inferiores aos resultados do IFbyGP, ficaram dentro de uma margem aceitável. Já nos experimentos com as alterações das resoluções (maiores), algumas das melhores sequências de operadores se mostraram ineficientes, muitas vezes encontrando outras formas que não eram esperadas dentro do contexto de pesquisa. Nos casos onde as sequências de operadores foram testadas pelo SHAIP-CGP sobre imagens com pequenas alterações nas suas formas originais, os resultados foram mais apropriados. Os resultados comprovam que o sistema proposto mostrou uma maior tendência para filtragem de imagens e reconhecimento de pequenos objetos com formas geométricas bem definidas.

## 6.3 Trabalhos Futuros

Como trabalhos futuros, pretende-se implementar: um algoritmo em FPGA visando-se o processamento rápido de imagens, uma vez que a notação compacta empregada pelo SHAIP-CGP permite uma implementação baseada em *hardware* reconfigurável. Também é pretendido modificar o sistema para processamento de imagens em tons de cinza com o intuito de comparar os resultados a serem obtidos com alguns trabalhos já publicados na literatura.

# REFERÊNCIAS


BARALDI, P. et al. Genetic algorithm-based wrapper approach for grouping condition-monitoring signals of nuclear power plant components. **Integrated Computer-Aided Engineering,** v. 18, n.3, p. 221-234, 2011.

BRADSKI, G. **The OpenCV Library**: Dr. Dobb's Journal of Software Tools, 2000.

DONGJIAN, X. et al. Application of genetic algorithms in remote sensing image processing. In: COMPUTATIONAL INTELLIGENCE AND SOFTWARE ENGINEERING, 2010. **Proceedings…**CiSE, 2010. p.1-3.

FACON, J. **Morfologia Matemática**: teoria e exemplos. Curitiba: Ed. Universitária Champagnat PUC-Paraná, 1996. 320 p.

FOGEL, D. **Evolutionary Computation**: toward a new philosophy of machine intelligence. Piscataway, NJ: IEEE Press, 1995.

FOGEL, L. J.; OWENS, A. J.; WALSH, M. J. **Artificial Intelligence through Simulated Evolution**. John Wiley and Sons, 1966.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 2. ed. Upper Saddle River, NJ: Prentice Hall, 2002. 204 p.

HARDING, S.; BANZHAF, W.; MILLER, J. F. A survey of self-modifying cartesian genetic programming. In: RIOLO, Rick; MCCONAGHY, Trent; VLADISLAVLEVA, Ekaterina (Ed.). **Genetic Programming Theory and Practice VIII**.Springer, 2010. p. 91-107 (Genetic and Evolutionary Computation, v. 8)

HARDING, S.; LEITNER, J.; SCHMIDHUBER, J. Cartesian genetic programming for image processing. In: RIOLO, Rick (Ed.) et al. **Genetic Programming Theory and Practice X**. New York: Springer, 2012. (Genetic and Evolutionary Computation).

HOLLAND, J. Outline for a logical theory of adaptive systems. **Journal of the ACM**, Ann Arbor, v. 9, n. 3, p. 297-314, 1962.

HOLLAND, J. Nonlinear environments permitting efficient adaptation. In: **Computer and Information Sciences II**. Academic Press, 1967.

HOLLAND, J. **Adaptation in Natural and Artificial Systems**. Ann Arbor, MI: University of Michigan Press, 1975.

JENNANE, R. et al. Genetic algorithm and image processing for osteoporosis diagnosis," Engineering in Medicine and Biology Society (EMBC). In: ANNUAL INTERNATIONAL CONFERENCE OF THE IEEE, 2010, Buenos Aires.

**Proceedings**… Buenos Aires: IEEE, 2010. p. 5597-5600. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5626804&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5626804>. Acesso em: 10 jan. 2013.

KOZA, J. R. **Genetic Programming:** on the Programming of Computers by Natural Selection. Massachusetts: MIT Press Cambridge, 1992.

LEITNER, J. et al. Mars terrain image classification using cartesian genetic programming. In: INTERNATIONALI SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, ROBOTICS AND AUTOMATION IN SPACE, 11., 2012. **Proceedings…** i-SAIRAS, 2012. Disponível em: < http://www.idsia.ch/~juergen/isairas2012.pdf>. Acesso em: 12 jan. 2013.

LI, Y.; QING-LAN, J. The application of improved evolutionary strategy algorithm in optimization. In: MACHINE LEARNING AND CYBERNETICS, 2012, Xian. **Proceedings**...Xian: IEEE, 2012. p.1212-1217. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6359528&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6359528>. Acesso em: 15 jan. 2013.

LIU, F.; YANG, B.; KAI, G. L. **The Bi-Group evolutionary programming for image processing,** 2012. In: INTERNATIONAL CONFERENCE ON AUDIO, LANGUAGE AND IMAGE PROCESSING, 2012, Shanghai. **Proceedings…** Shanghai: IEEE, 2012. p. 832-836. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6376729>. Acesso em: 04 abr. 2013.

MATHERON, G. **Random sets and integral geometry**. New York: John Wiley and Sons, 1975. 261 p.

MATTHEWS, B. W. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. **Biochimica et Biophysica Acta**, v. 405, n. 2, p. 442–451, 1975.

MILLER, J. F.; THOMSON, P.; FOGARTY, T. C. Designing electronic circuits using evolutionary algorithms. Arithmetic Circuits: a case study. In: QUAGLIARELLA, D. (Ed.). **Genetic Algorithms and Evolution Strategies in Engineering and Computer Science.** Chichester: John Wiley and Sons, 1997.

MILLER, J. F. An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, 1999. **Proceedings…** 1999. p. 1135-1142. Disponível em:<http://www.cartesiangp.co.uk/papers/gecco1999b-miller.pdf>. Acesso em 20 ago. 2011.

MILLER, J. F.; THOMSON, P. Cartesian Genetic Programming. In: ____. **Genetic Programming:** European Conference, EuroGP 2000, Edinburgh, Scotland, UK, April 15-16, 2000. Proceedings. New York: Springer Berlin Heidelberg, 2000. p. 121-132. Disponível em: <http://link.springer.com/chapter/10.1007%2F978-3-540-46239-2_9>. Acesso em: 22 ago. 2011. (Lecture Notes in Computer Science, v. 1802)

MILLER, J. F.; SMITH, S. L. Redundancy and Computational Efficiency in Cartesian Genetic Programming. **IEEE Transactions on Evolutionary Computation,** v. 10, n.2, p. 167-174, 2006.

MILLER, J. F. (Ed.). **Cartesian Genetic Programming.** Berlin: Springer-Verlag 2011. p. 365. (Natural Computing Series)

MILLER, J.F. **GECCO 2013 tutorial:** cartesian genetic programming. GECCO, 2013. 715-740 p.

MONTES, H. A.; WYATT, J. L. Cartesian genetic programming for image processing tasks. In: INTERNATIONAL CONFERENCE ON NEURAL NETWORKS AND COMPUTATIONAL INTELLIGENCE, 2003. **Proceedings....** Birmingham: IASTED, 2003. p.185-190. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.2243&rep=rep1&type=pdf>. Acesso em: 25 jan. 2013.

PEDRINO, E. C. **Arquitetura pipeline reconfigurável através de instruções geradas por programação genética para processamento morfológico de imagens digitais utilizando FPGA's**. 2008. 220 p. Tese (Doutorado em Engenharia Elétrica) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

PEDRINO, E. C. et al. Automatic construction of image operators using a genetic programming approach. In: INTERNATIONAL CONFERENCE ON INTELLIGENT SYSTEMS DESIGN AND APPLICATIONS (ISDA), 11., 2011, Córdoba. **Proceedings…** Córdoba: IEEE, 2011. p. 636-641. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6121727>. Acesso em: 09 mai. 2012.

PEDRINO, E. C. et al. A genetic programming based system for the automatic construction of image filters. **Integrated Computer-Aided Engineering**, Brazil, v. 20, n. 3, p. 275-287, 2013.

RECHENBERG, I. *Cybernetic solution path of an experimental problem.* UK: Ministry of Aviation. Royal Aircraft Establishment, 1965.

RECHENBERG, I. **Evolutions strategie:** optimierung technischer systeme nach prinzipien der biologischen evolution. Stuttgart: Fommann-Holzboog, 1973.

REHMAN, A.; KHAN, G. M. Polymorphic Circuit Design for Speedy Handwritten Character Recognition Using Cartesian Genetic Programming. In: FRONTIERS OF INFORMATION TECHNOLOGY. **Proceedings...** Islamabad: IEEE, 2011. p. 79-84. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6137123>. Acesso em: 08 mai. 2013.

SÁNCHEZ-MONTERO, R. et al. Efficient design of a double-band coplanar hybrid antenna using multi-objective evolutionary programming. **International Journal of**

**Numerical Modelling:** Electronic Networks, Devices and Fields, v. 26, n. 6, p. 620-629, 2012.

SCHWEFEL, H-P. **Numerical optimization of computer models**. Chichester: John Wiley and Sons, 1981.

SPEARS, W. M. et al. An overview of evolutionary computation. In: ____, **Machine Learning: ECML-93:** European Conference on Machine Learning Vienna, Austria, April 5-7, 1993 Proceedings. New York: Springer Berlin Heidelberg, 1993. p. 442-459 (Lecture Notes in Computer Science, v. 667).
Disponível em: <http://link.springer.com/chapter/10.1007/3-540-56602-3_163>. Acesso em 07 nov. 2012.

WANG, J.; CHEN, Q. S.; LEE, C. H. Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware, **IET Comput. Digit. Tech.**, Incheon, v. 2, n. 5, p. 386-400, set. 2008.

XIAODONG, Y.; JINGBO, S.; HONGBIN, D. On evolutionary strategy based on hybrid crossover operators. In: INTERNATIONAL CONFERENCE ON ELECTRONIC AND MECHANICAL ENGINEERING AND INFORMATION TECHNOLOGY, v. 5, 2011, Harbin. **Proceedings...** Harbin: IEEE, 2011. p. 2355-2358, Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6023583&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6023583>. Acesso em: 06 feb. 2013.