UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RAFAEL FENSTERSEIFER

Aplicação de Algoritmos Genéticos na redução de ruído em imagens

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Dante Barone

Porto Alegre
2014

RESUMO

A aplicação de Algoritmos Genéticos na busca por soluções têm se mostrado eficiente em diversas áreas, entre elas o processamento de imagens. Inevitavelmente, no processo de captura, transmissão e gravação/leitura de imagens, surgem imperfeições como o caso do ruído. Por representar um um problema de extrema importância, tanto visualmente para um usuário que use uma câmera fotográfica como para processos que dependem de reconhecimento de imagens, é necessário que sejam desenvolvidos filtros cada vez mais eficientes para a remoção do ruído em imagens. Outros fatores também têm se mostrado importantes no desenvolvimento de filtros, como a necessidade de que sejam implementáveis em hardware com pouco uso de recursos e que respeite a diversidade existente entre tantos meios onde é utilizado. Nesse trabalho é apresentado um estudo sobre a viabilidade e eficiência do uso de Algoritmos Genéticos no desenvolvimento de filtros para remoção de ruído em imagens. A partir de um banco de imagens de referência, e sem a necessidade de qualquer tipo de desenvolvimento ou intervenção humana, filtros são gerados para a remoção do ruído presente nas imagens.

Palavras-chave: Algoritmos Genéticos, Filtro de ruído

Application of Genetic Algorithm in image noise reduction

ABSTRACT

The application of Genetic Algorithm in the search for solutions has proven to be efficient in many areas, including image processing. Inevitably, in the image capture, transmission and saving/reading process, some imperfections arise, such as noise. As it represents a problem of extreme importance visually, for a user of photographic cameras, and for the processes that depend of image recognition, it is necessary to develop more efficient filters for image noise removal. Other factors have also shown to be important in the development of filters, like the necessity to be implementable in hardware with little use of resources and that respects the existing diversity among the many means where it is used. In this paper, a study about the viability and efficiency of the use of Genetic Algorithms in the development of filters for the removal of image noise is presented. Using an image bank as reference and without the necessity of any kind of human intervention or development, filters are generated in order to remove the noise present in the images.

Keywords: Genetic Algorithm, Noise filter

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmos Genéticos
CGP	Cartesian Genetic Programming
MF	Common Median Filter
CWMF	Center Weighted Median Filter
WMF	Weighted Median Filter
AMF	Adaptative Median Filter
PWMAD	Pixel-Wise Median of the Absolute Deviations from de median
DWM	Directional Wighted Median filter

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais a computação tem alcançado importância elevada em diversos processos. Em particular, o processamento de imagens se torna indispensável em diversas áreas que vão desde o entretenimento até o controle de processos industriais. Juntamente com a captura e transmissão de imagens anda o ruído que pode ser gerado por diversos fatores. Sendo uma característica inevitável, é indispensável que se busquem filtros cada vez melhores que tenham a capacidade de remover as imperfeições, mantendo toda a informação presente na imagem, bem como seus detalhes. O grande entrave é justamente o fato de segmentos tão distintos terem necessidades muito diferentes entre si e mesmo assim necessitarem de filtros eficientes.

Nesse contexto surge a proposta da fusão de dois conceitos, o processamento de imagens e a Computação Evolutiva. Como demostrado no trabalho, muitos estudos têm sido feitos sobre o tema em diversas frentes e os resultados têm se mostrado bastante animadores, o que demonstra ser uma área promissora. Esse trabalho tem como proposta o estudo sobre filtros para remoção de ruído de imagens e a possibilidade do emprego de Algoritmos Genéticos para a síntese totalmente autônoma de algoritmos eficientes a partir de imagens de referência.

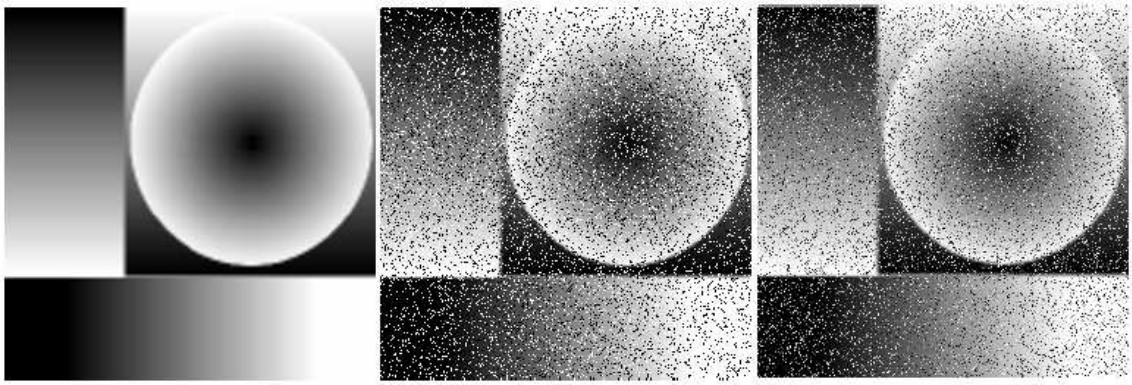
No capítulo 2 é apresentado um estudo sobre os diferentes tipos de ruído presentes em imagens e no capítulo 3, um estudo sobre os filtros utilizados para sua remoção. No capítulo 4 é introduzido o conceito a se utilizar no projeto. Nos capítulos 5, 6 e 7 é apresentada a idéia do algoritmo genético para solução do problema, o ambiente de desenvolvimento utilizado e como foi o processo de desenvolvimento. Ao final, nos capítulos 8 e 9, a apresentação dos resultados obtidos e discussões finais sobre estes e propostas para trabalhos futuros.

2 INTRODUÇÃO AO RUÍDO EM IMAGENS

Em imagens, ruído pode ser definido como alterações na intensidade e cor de um pixel com relação ao que está sendo capturado ou transmitido originalmente. Essas deformidades podem surgir por diversos fatores externos como fatores físico-elétricos em sensores digitais de captura, grãos presentes em filmes fotográficos ou mesmo por falhas na transmissão ou leitura/gravação.

Como descrito em Digital Imagen Processing, “ruído proveniente de sensores ruidosos ou erros de canais de transmissão normalmente aparecem de forma discreta e com variações isoladas sem correlação espacial. Pixels com defeito, visivelmente são diferentes de seus vizinhos” (PRATT, 2007, p. 267). Simplificadamente, ruídos são divididos em dois tipos básicos, de saturação e variação. No primeiro, o valor dos pixels atingidos é alterado para seu valor máximo ou mínimo, sem variações intermediárias. São mais conhecidos como ruído de sal-e-pimenta. Esse tipo é muito utilizado didaticamente por sua simplicidade, mas na prática são raros de acontecer. O segundo, comumente encontrado em casos reais, pode atingir um pixel com maior ou menor intensidade, efetuando mudanças sensíveis ou drásticas em seu valor. É mais difícil de ser detectado em imagens. Na Figura 2.1 são apresentados exemplos desses diferentes tipos de ruído:

Figura 2.1 – Exemplo de ruído sal-e-pimenta e ruído de intensidade



Fonte: Evolutionary Design ofRobust Noise-Specific Image Filters (2011)

Além de aspectos visuais, onde o usuário de câmera fotográfica deseja que suas fotos sejam coerentes com a realidade e sem inperfeições, ruído é extremamente importante para aplicações baseados em processamento de imagens. Segmentação e

identificação de padrões são comumente utilizados na indústria para a automação de linhas de produção utilizando robótica, por exemplo. Filtrar imagens retirar o ruído faz parte de uma fase chamada pré-processamento de imagens que é crucial para o bom funcionamento do sistema. Sem tal processo, os algoritmos empregados nas fases seguintes se tornam muito menos precisos e podem até parar de funcionar.

Pela importância apresentada, a remoção de ruído tem sido muito estudada ao longo dos anos e deve ser considerada quando no desenvolvimento de aplicações reais. Para testar a eficiência de filtros, é necessário que se tenha uma imagem de referência além da imagem ruidosa para que se possa fazer a comparação dos resultados. Normalmente o processo consiste em utilizar uma imagem limpa como referência e gerar a imagem ruidosa artificialmente através de alguma fórmula de distribuição. O Matlab, por exemplo, possui diversas fórmulas para criar sinteticamente diversos tipos de ruídos com distribuição e intensidade diferentes em cada caso, como distribuições uniformes, gasussianas e exponenciais. Na Figura 2.2 é apresentada uma tabela com várias formulações disponíveis no Matlab para geração de ruído em imagens:

Figura 2.2 – Funções do Matlab utilizadas para geração de ruído

TABLE 5.1 Generation of random variables.

Name	PDF	Mean and Variance	CDF	Generator†
Uniform	$p_z(z) = \begin{cases} \frac{1}{b-a} & \text{if } a \le z \le b \\ 0 & \text{otherwise} \end{cases}$	$m = \frac{a+b}{2}, \quad \sigma^2 = \frac{(b-a)^2}{12}$	$F_z(z) = \begin{cases} 0 & z < a \\ \frac{z-a}{b-a} & a \le z \le b \\ 1 & z > b \end{cases}$	MATLAB function `rand`
Gaussian	$p_z(z) = \frac{1}{\sqrt{2\pi}b} e^{-(z-a)^2/2b^2}$ $-\infty < z < \infty$	$m = a, \quad \sigma^2 = b^2$	$F_z(z) = \int_{-\infty}^{z} p_z(v)\, dv$	MATLAB function `randn`
Salt & Pepper	$p_z(z) = \begin{cases} P_a & \text{for } z = a \\ P_b & \text{for } z = b \\ 0 & \text{otherwise} \end{cases}$ $b > a$	$m = aP_a + bP_b$ $\sigma^2 = (a-m)^2 P_a + (b-m)^2 P_b$	$F_z(z) = \begin{cases} 0 & \text{for } z < a \\ P_a & \text{for } a \le z < b \\ P_a + P_b & \text{for } b \le z \end{cases}$	MATLAB function `rand` with some additional logic
Lognormal	$p_z(z) = \frac{1}{\sqrt{2\pi}bz} e^{-[\ln(z)-a]^2/2b^2}$ $z > 0$	$m = e^{a+(b^2/2)}, \sigma^2 = [e^{b^2} - 1]e^{2a+b^2}$	$F_z(z) = \int_0^z p_z(v)\, dv$	$z = ae^{bN(0,1)}$
Rayleigh	$p_z(z) = \begin{cases} \frac{2}{b}(z-a)e^{-(z-a)^2/b} & z \ge a \\ 0 & z < a \end{cases}$	$m = a + \sqrt{\pi b/4}, \sigma^2 = \frac{b(4-\pi)}{4}$	$F_z(z) = \begin{cases} 1 - e^{-(z-a)^2/b} & z \ge a \\ 0 & z < a \end{cases}$	$z = a + \sqrt{-b \ln[1 - U(0,1)]}$
Exponential	$p_z(z) = \begin{cases} ae^{-az} & z \ge 0 \\ 0 & z < 0 \end{cases}$	$m = \frac{1}{a}, \quad \sigma^2 = \frac{1}{a^2}$	$F_z(z) = \begin{cases} 1 - e^{-az} & z \ge 0 \\ 0 & z < 0 \end{cases}$	$z = -\frac{1}{a}\ln[1 - U(0,1)]$
Erlang	$p_z(z) = \frac{a^b z^{b-1}}{(b-1)!} e^{-az}$ $z \ge 0$	$m = \frac{b}{a}, \quad \sigma^2 = \frac{b}{a^2}$	$F_z(z) = \left[1 - e^{-az} \sum_{n=0}^{b-1} \frac{(az)^n}{n!}\right]$ $z \ge 0$	$z = E_1 + E_2 + \cdots + E_b$ (The E's are exponential random numbers with parameter a.)

† $N(0, 1)$ denotes normal (Gaussian) random numbers with mean 0 and a variance of 1. $U(0, 1)$ denotes uniform random numbers in the range $(0, 1)$.

Fonte: Digital Image Processing using Matlab (2004, p. 146)

Além disso, certos cenários possibilitam a implementação de outras alternativas muito mais próximas da realidade. Não necessariamente imagens ruidosas devem ser

sintéticas para se ter uma imagem limpa. Existem técnicas que possibilitam gerar uma imagem muito bem filtrada a partir da sobreposição de várias imagens ruidosas do mesmo assunto. Dessa forma, não é necessário a aplicação de filtros estatístico, mantendo grande quantidade de detalhes preservados. Na Figura 2.3 um exemplo dos resultados da aplicação dessa técnica:

Figura 2.3 – Remoção de ruído pela sobreposição de várias imagens

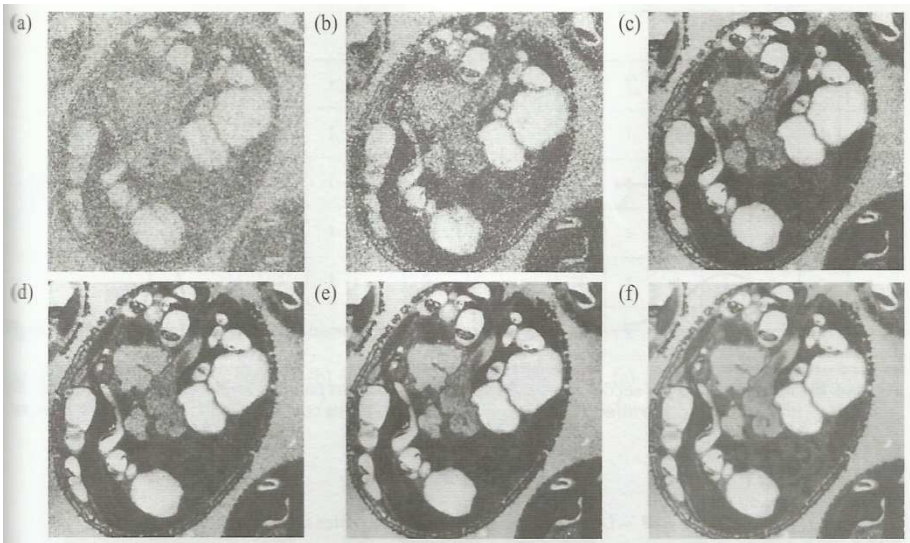


Figura 4.18 *— Exemplo de redução de ruído através de médias: (a) uma imagem ruidosa típica; (b)-(f) resultados da média de 2, 8, 16, 32 e 128 imagens ruidosas.*

Fonte: Processamento de Imagens Digitais (2000, p. 135)

3 INTRODUÇÃO ÀS TÉCNICAS DE FILTRAGEM EM IMAGENS

Para a remoção do ruído das imagens, foram desenvolvidos ao longo do tempo diversos filtros com enfoques e técnicas muito diferentes entre si. Cada um desses apresenta vantagens e desvantagens com relação a diversos parâmetros de análise. Uns são extremamente eficientes para determinado tipo de ruído enquanto apresentam péssimos resultados para outros tipos. Quantidade de memória utilizada no processo e tempo de processamento também podem ser fatores cruciais em determinadas aplicações. Existem hoje duas grandes famílias de filtros, os lineares e os não lineares.

Filtros Lineares

Amplamente utilizados para a remoção de ruído pois são extremamente simples. São basicamente compostos de uma matriz em que é feita a convolução com a imagem original para ser gerada uma nova imagem filtrada. Na prática, nada mais é que um filtro passa-baixa, removendo pixels completamente isolados que geram altas frequências. A desvantagem do sistema é que gera borramento da imagem, independente do tipo de ruído e de sua intensidade. As vantagens são a velocidade de aplicação e a possibilidade de modelagens matemáticas altamente robustas em seu desenvolvimento.

Filtros não lineares

Justamente pelo fato do sinal de ruído, na prática, não ter comportamento linear, métodos não lineares apresentam resultados melhores.

Filtros de médias e medianas

Atualmente são os filtros mais utilizados por apresentarem resultados satisfatórios sem a necessidade de grande quantidade de memória e poder de processamento. Ao invés de realizar convolução a partir de uma matriz em um determinado pixel e seus vizinhos, calcula a média ou mediana dessa vizinhança e determina o valor final deste. Alguns métodos ainda utilizam uma análise de contexto maior, fazendo um levantamento estatístico de toda a imagem para ser utilizada no cálculo local. Nesse contexto os filtros mais conhecidos são Common Media Filter (MF), Center Weighted Median Filter (CWMF), Weighted Median Filter (WMF).

Detecção de ruído

Mesmo que o efeito de borramento seja muito menos perceptível nesse tipo de filtro, ainda ocorre e há perda de detalhes importantes, pois são aplicados a todos os pixels da imagem. Para resolver esse problema, surgiu um novo conceito de filtragem em que o processo é dividido em duas fases. Em primeiro lugar o objetivo é identificar quais pixels da imagem são afetados por ruído para então aplicar algum método conhecido para substituir o valor errado por uma aproximação. Diferentemente dos outros métodos, a fase de detecção utiliza filtros de ordem estatística que aumentam gradativamente sua máscara na procura pelo melhor resultado. Essa abordagem aumenta consideravelmente a capacidade do filtro manter os detalhes da imagem. Esse é o conceito utilizado pelo método Adaptative Median Filter (AMF).

Métodos iterativos

Na busca por melhores resultados surgiram métodos iterativos, que tentam determinar com melhor precisão se os pixels são ou não ruído e como calcular seu valor com melhor precisão. Para isso, entre outras técnicas, procura por arestas e outras formas de contextualização. Dentre os algoritmos que utilizam essas técnicas estão Pixel-Wise Median of the Absolute Deviations from de median (PWMAD) e Directional Wighted Median filter (DWM). A desvantagem é clara, por se tratar de métodos iterativos, a quantidade de memória e capacidade de processamento é extremamente maior que os demais métodos. Mesmo assim são considerados o estado da arte por apresentarem os melhores resultados até hoje.

4 DEFINIÇÕES INICIAIS DE PROJETO

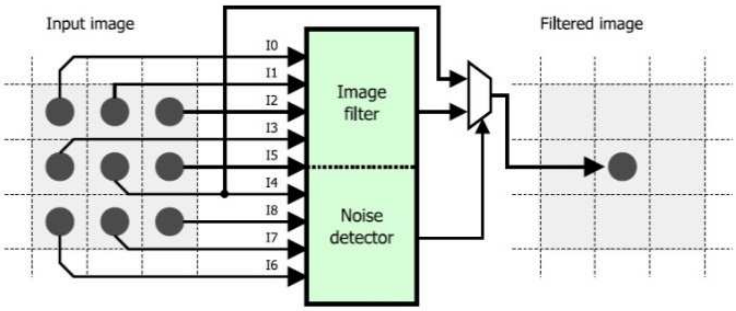
A partir do estudo realizado e antes de qualquer estudo de implementação, é necessário determinar quais serão os critérios de projeto utilizados para que se possa determinar uma modelagem do sistema pretendido.

Como citado no capítulo inicial, aplicações industriais são um ótimo exemplo da necessidade de aplicação de filtros no pré-processamento das imagens capturadas para o controle da linha de produção. Aplicações reais como essas necessitam de cálculos extremamente eficientes em velocidade, confiabilidade, etc., pois a atuação de mecanismos de controle dependem disso. Outro exemplo é o caso do fotógrafo que quer ver o resultado de sua captura logo após o clique da máquina. Nesse caso, além de velocidade o tamanho do hardware é limitado. Esses fatores fazem com que em muitos casos o circuito de filtragem não seja simplemente um software, mas que possa ser implementado em hardware para que respeite as decisões de projeto de cada aplicação. Independente do caso, o foco do trabalho é dimensionar algoritmos que possam ser implementados em hardware simples, como FPGAs por exemplo, sem a necessidade de hardware extremamente dispendioso pelo fato de ser necessário quantidades enormes de memória ou processadores dedicados a isso.

A partir do primeiro item, podemos fazer uma nova análise das técnicas de filtragem estudadas. Métodos iterativos são extremamente eficientes mas vão contra a primeira definição. Dentre os métodos restantes, o conceito que apresentou grande avanço na remoção de ruído foi de separar o processo em duas etapas – detecção de ruído e cálculo do novo valor de cada pixel. Portanto, um circuito que representa tal problema deve ter como entrada uma certa quantidade de pixels e dois valores de saída, um indicando se o pixel analizado é ruído e outra deterrminando um valor calculado por um filtro. Caso o valor da detecção seja falso, o pixel base de entrada é simplesmente colocado na saída. Caso contrário, o pixel calculado pelo filtro deve ser utilizado.

Combinando essas informações, podemos fazer um modelo esquemático de como deve ser o circuito responsável pela aplicação do filtro nas imagens. Na figura 4.1 podemos ver um diagrama representando este:

Figura 4.1 – Proposta de circuito para remoção de ruído em imagens



Fonte: Evolutionary Design ofRobust Noise-Specific Image Filters (2011)

O ruído pode aparecer nas imagens apresentando diferentes tipos de padrões em função de origem do mesmo. Sensores de câmeras fotográficas apresentam deformações diferentes das apresentadas em erros na transmissão que, por sua vez, podem ser diferentes do ruído presente em imagens digitais de raios X ou tomografia computadorizada. Isso faz com que cada segmento possua necessidades diferentes e consequentemente uma implementação com características próprias. Para que isso seja colocado em prática, é necessário que se faça um estudo de cada caso, se encontre a melhor solução existente para o caso ou se implemente uma nova. Claramente esse é um processo dispendioso, pois será necessário para cada caso fases de estudo, implementação e testes. Em muitos casos a solução é utilizar as ferramentas conhecidas mesmo que não forneçam resultados ótimos.

Na busca por um filtro próprio para cada situação são possíveis duas abordagens. Na primeira um filtro conhecido é escolhido. A partir disso o desenvolvimento se resume a acertar os parâmetros desse para que se ajuste da melhor forma para produzir os melhores resultados. A outra alternativa é iniciar o processo de desenvolvimento desde o começo e produzir um filtro totalmente novo.

5 ALGORITMOS GENÉTICOS NO PROCESSAMENTO DE IMAGENS

O problema descrito no capítulo anterior é recorrente e está presente em diversas áreas do processamento de imagens. Desde fases de pré-processamento, como filtros para remoção de ruído e ajustes de contraste, até pós-processamento, como compactação de imagens para transmissão. Na busca por soluções cada vez mais específicas, diversos estudos têm sido feitos utilizando Algoritmos Genéticos para encontrar tais soluções. Os resultados encontrados são muitos animadores e a comunidade científica tem publicado diversos artigos conprovando a eficácia de tal relacionamento. Exemplos são: Application of Image Segmentation Algorithm Based on Particle Swarm Optimization and Rough Entropy Standard, Discrete Cosine Transform Image Compression Based on Genetic Algorithm, Automatic Design of Image Operators using Evolvable Hardware e Evolutionary Design of Robust Noise-Specific Image Filters.

Além de os resultados apresentados serem comparáveis aos métodos conhecidos, essa abordagem apresenta a grande vantagem de encontrar a melhor solução a partir de dados de entrada conhecidos. O desenvolvedor da solução não precisa mais se preocupar em pesquisar e desenvolver algoritmos novos para cada caso, basta que forme um bom conjunto de imagens de entrada para que o AG seja treinado corretamente para o contexto em que se deseja que atue. Utilizando alguma das técnicas descritas anteriormente, deve-se possuir uma imagem limpa e uma imagem defeituosa. A partir da comparação entre as duas, a evolução da solução ocorre com a busca de minimização entre o resultado da aplicação de um candidato na imagem com defeito e a imagem de referência.

Até alguns anos, uma abordagem muito utilizada foi de gerar um circuito em árvore. Nessa abordagem uma série de entradas era consecutivamente reduzida por funções até que no final restasse o número de saídas determinadas. Um exemplo de redução é, a cada duas entradas, gerar uma saída utilizando uma função determinada (como soma, por exemplo). Novos trabalhos têm sido feitos utilizando uma nova técnica conhecida como Cartesian Genetic Programming (CGP). Essa técnica têm se mostrado muito eficiente pelo fato de, diferentemente do conceito de árvore, permitir que caminhos alternativos sejam criados ao longo das gerações e acabem agregando

possibilidades de funcionalidades e caminhos maiores aos resultados. Esse fenômeno é descrito pelo autor do artigo Cartesian Genetic Programming como redundância.

Um estudo que marca a utilização desse conceito está descrito no artigo Gate-Level Optimization of Polymorphic Circuits Using Cartesian Genetic Programming. Simplificadamente, o problema descrito trata da síntese de circuitos digitais complexos. Esses circuitos fazem uso de portas lógicas com comportamento dependente de fatores externso, como temperatura por exemplo. Uma determinada porta pode gerar resultados verdade como uma porta NAND se o componente se encontra abaixo de uma determinada temperatura. Caso contrário funciona como uma NOR. Com isso é possível economizar em componentes, e consequentemente em espaço, para que o circuito tenha comportamento específico para cada faixa de temperatura. Entretanto, os softwares atuais não têm se mostrado eficientes para executar a síntese de tais circuitos. A solução encontrada foi utilizar o CGP para executar essa síntese a partir de tabelas verdade com o comportamento desejado.

No processamento de imagens também tem sido amplamente utilizado pelo fato de que o processamento não deixa de ser um circuito onde algumas entradas são analisadas, como um pixel referência e alguns vizinhos, e a partir disso gera uma saída que é analisada com alguma referência. O objetivo é que se minimize a diferença até que os resultados sejam satisfatórios.

O termo cartesiano provavelmente tenha surgido porque o conceito é baseado em uma matriz de *m* linhas por *n* colunas que são referenciadas cartesianamente por coordenadas de duas dimensões. Cada posição dessa matriz representa uma célula que pode ser ocupada por uma das funções pré-definidas e ter as entradas ligadas às saídas de outras posições ou diretamente às entradas do circuito.

Cada fonte de dados do circuito é numerado com um valor inteiro partindo do valor zero. Uma fonte de dados podem ser as entradas principais, que recebem os dados de entrada do circuito, e as saídas de cada uma das células. A numeração começa nas entradas principais e continua a contagem com as saídas das células linha a linha, de cima para baixo, em cada uma das colunas.

Definições

Partindo do conceito geral de um candidato, surge a necessidade de definir e nomear diversos parâmetros que podem alterar drasticamente o comportamento e as possibilidade de um candidato.

- o circuito geral pode ter nr linhas por nc colunas de células programáveis

- o circuito pode ter pi entradas e po saídas

- cada célula pode ter ni entradas e no saídas

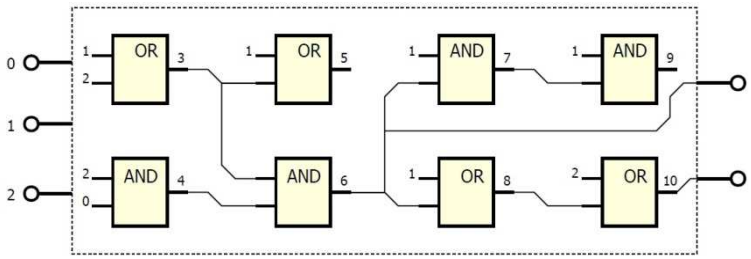
- as entradas de cada célula só podem ser conectadas com saídas de células de colunas anteriores ou com as entradas principais. O parâmetro *l* determina o número de colunas anteriores em que essa conexão pode ser feita. No caso de $l = 1$, somente colunas vizinhas podem ser conectadas. Ao contrário, se $l = nc$, a conectividade é total e uma saída por ser ligada diretamente com uma entrada.

Genótipo

Cada candidato a solução é representado por uma sequência de números inteiros de tamanho fixo, onde são listadas as conexões de entrada de cada uma das *nr* x *nc* células mais a conexão de cada uma das saídas do circuito. Cada célula é representada por uma tupla com *ni* valores representando as conexões de entrada mais um número representando a função do nodo. Portanto, um genótipo completo é formado por uma sequência de $(ni + 1).nc.nr + po$ valores inteiros.

Para exemplificar esses conceitos, podemos ver Na Figura 5.1 um exemplo de circuito candidato para uma configuração onde $nr = 2$, $nc = 4$, $pi = 3$, $po = 2$ e $ni = 2$ e $no = 1$:

Figura 5.1 – Exemplo de circuito em CGP



Fonte: Evolutionary Design ofRobust Noise-Specific Image Filters (2011)

Nesse exemplo foram definidas duas funções possíveis: *AND*, com valor zero, e *OR*, com valor um. O genótipo resultante seria: 1,2,1, 2,0,0, 1,3,1, 3,4,0, 1,6,0, 1,6,1, 1,7,0, 2,8,1, 6, 10. Cada tupla e valores de saída foram separados por um espaço adicional para facilitar a visualização.

Processamento de imagens

Cada pixel de uma imagem é representado por valores inteiros de oito bits, ou seja, podem assumir valores na faixa entre 0 e 255. Para simplificar o projeto é utilizada apenas uma camada de cor no processamento da imagem (preto-e-branco). O aprimoramento para efetuar o tratamento multi-camadas se torna bastante simples, uma vez que a primeira etapa tenha sido concretizada.

Nesse contexto, as entradas do circuito devem ser o pixel principal a ser analisado e certa quantidade de pixels vizinhos. Essa quantidade pode ser de nove, em que são utilizados oito vizinhos, formando uma matriz de três por três pixels; vinte e cinco, onde são selecionados cinco por cinco pixels, e assim por diante. Cada entrada representa oito bits, podendo assumir valores entre zero e duzentos e cinquenta e cinco.

Pra manter a compatibilidade total entre todas as partes do circuito e a saída, optou-se por não utilizar uma saído boolenada para a decisão de determinado pixel ser ruído ou não. Para tanto, é utilizado o valor do bit mais significativo da saída, portanto valores maiores que cento e vinte e sete representa o valor verdadeiro e valores menores que isso representam o valor falso. A outra saída representa os oito bits do pixel de saída do filtro.

6 DESCRIÇÃO DA FERRAMENTA UTILIZADA NA IMPLEMENTAÇÃO

Na implementação da solução foi utilizado o software MatLAB, pois esse apresenta pacotes para manipulação de imagens e, principalmente, por proporcionar um pacote totalmente funcional para modelagem de problemas envolvendo algoritmos genéticos. A primeira barreira encontrada foi o fato de somente a partir da versão 2012b o pacote suportar problemas descritos somente utilizando variáveis inteiras, uma vez que originalmente o pacote foi concebido para minimização de problemas matemáticos de ponto flutuante.

Matlab GA

O pacote de Algoritmos Genéticos do programa funciona da maneira convencional como o assunto é apresentado nas cadeiras de Inteligência Artificial do Instituto de Informática. O objetivo básico é minimizar o valor do fitness dos indivíduos. Em primeiro lugar é gerada a população inicial (normalmente com valores totalmente aleatórios). Em seguida inicia o processo que é utilizado até que se atinja um dos critérios de parada, como segue:

- seleciona os pais da nova geração a partir de seus valores de fitness (por padrão, o programa faz essa seleção utilizando como critério a seleção dos elementos com menor valor de fitness)

- os candidatos com menor fitness da população são selecionados como elite e serão passados para a nova população - por padrão cerca de 5% dos elementos são preservados

- os elementos selecionados como pais têm alguns de seus valores alterados (mutação) e são combinados para gerar novos filhos (crossover) – por padrão cerca de 1% dos valores sofrem mutação e 80% da nova geração são oriundos de crossover

- os novos filhos são colocados no lugar da população atual para formar a nova geração

A implementação do pacote para problemas puramente inteiros sofreu algumas alterações com relação ao algoritmo básico para manter a integridade e impôs algumas restrições ao usuário. Funções de geração da população inicial, mutação e crossover especiais foram criadas para garantir que o resultado das operações resultasse somente

em números inteiros. O valor a ser minimizado é chamado de função de penalti, que é determinante para a seleção dos elementos da nova população via torneio. Esse valor é determinado como segue: no caso do candidato ser válido (respeitar as restrições), é assumido o valor de seu fitness; caso contrário, o valor é igual ao pior fitness mais uma constante (por padrão 100).

Implementação

Para a manipulação de candidatos, foi gerada uma classe chamada rfCandidate (toda a nomeclatura começa com *rf* para diferenciação de nomes eventualmente pré-existentes). Essa classe pode ser construída zerada ou pode-se passar um genótipo para que seus valores sejam preenchidos a partir disso. Da mesma forma, um candidato tem a habilidade de calcular e retornar seu genótipo. Na prática, a classe representa o fenótipo e a sequência de inteiros representa o genótipo.

Parte importante no processo é a definição dos limites superior e inferior de cada um dos valores envolvidos no genótipo. Para isso são geradas duas sequências de inteiros com o mesmo tamanho do genótipo. Os valores de entradas das tuplas e os valores de saída po tem seus valores calculados a partir do valor definido para o parâmetro *l*, enquanto os valores que definem a função da célula dependem da quantidade de funções definidas.

O valor de saída pode ser calculado a partir da classe de representação do candidato, passando os valores de entrada. Para calcular a saída, cada célula é percorrida recursivamente através das conexões entre saídas e entradas até que se chegue nos valores *pi*. Como uma mesma célula pode ser alcançada por vários caminhos do circuito, seu valor é salvo temporariamente. Toda vez que uma entrada nova for utilizada como parâmetro para o circuito, os valores intermediários devem ser zerados.

Função de fitness

A função a ser minimizada nada mais é do que uma comparação pixel a pixel entre a imagem limpa e a imagem resultante da operação de aplicação do filtro gerado por um candidato. Cada pixel é comparado com o pixel resultante e o valor absoluto da diferença é adicionado a um somatório que corresponde ao valor final de fitness. Para ser analisado pelo circuito cada pixel necessita de certa quantidade de vizinhos, o que gera uma borda de pixels não analisado na imagem. Isso não se torna um problema, uma

vez que para a plicação final do filtro em imagens basta que se execute alguma função conhecida de padding. Abaixo a fórmula que representa o fitness de uma imagem sendo filtrada com um circuito que recebe uma matriz de 3 x 3 pixel como entrada:

Figura 6.1 – Fórmula para cálculo do fitness de um candidato

$$fit(If, Ir) = \sum_{i=2}^{R-1} \sum_{j=2}^{C-1} |If(i,j) - Ir(i,j)|$$

Fonte: Criado pelo autor

onde *If* representa a imagem filtrada e *Ir* a imagem de referência. *R* e *C* representam o número de linhas e colunas da imagem. No Matlab a indexação de linhas e colunas é feito com base no valor 1.

7 DESCRIÇÃO DO PROCESSO DE DESENVOLVIMENTO DA SOLUÇÃO

O processo de desenvolvimento da solução ocorreu de forma incremental. A cada etapa foram adicionadas funcionalidades e testadas para as suas funções. Inicialmente o objetivo foi carregar e salvar imagens, converter uma imagem colorida para preto-e-branco, gerar ruído sinteticamente e apresentar os resultados. Isso envolveu o uso do pacote de processamento de imagens do Matlab. Em seguida iniciou-se o desenvolvimento em si.

Em primeiro lugar, foi gerada a classe que representaria um candidato e seria responsável, em um primeiro momento, por gerenciar os dados relativos ao circuito e efetuar a conversão entre genótipo e fenótipo. Para tal, os membros da classe foram definidos como: uma tabela de tamanho *nr* x *nc* x *ni*+1, responsável por registrar cada tupla de origem das entradas e a função de cada célula, e um vetor com tamanho igual ao número de saídas, para registrar as conexões das mesmas. O construtor da classe, portanto, deve receber como argumentos *nr*, *nc*, *ni* e *po* (número de linhas, número de colunas, número de entradas por célula e número de saídas). Opcionalmente, o construtor pode receber como argumento um genótipo. Nesse caso o valor passado em *po* é ignorado, pois a partir dos outros argumentos e pelo tamanho da lista de inteiros sabe-se o número de saídas pela quantidade de argumentos excedentes.

A geração do genótipo é simples, a classe percorre cada um dos elementos da tabela de conexões e adiciona as tuplas em um vetor de inteiros. Ao final, adiciona as conexões das saídas principais. Adicionalmente, foi gerada uma função de apoio que retorna o tamanho do genótipo, ou seja, o número de valores que serão retornados. Além disso, é preciso gerar dois vetores de valores inteiros para indicar ao Matlab os limites inferiores e superiores de cada uma das posições dos valores do genótipo para que novos candidatos sejam validados no processo hereditário. Esses vetores têm tamanhos iguais ao tamanho do genótipo. Nas posições que indicam entradas de células os valores devem ser limitados às saídas de colunas anteriores e respeitar o valor de *l* indicado. Para as posições onde são definidas as funções das células, os valores devem ser limitados ao número de funções definidas.

Concluída a etapa de controle de dados, passou-se à fase de execução do circuito. Em primeiro lugar surgiu uma função para determinar a partir de um índice de

origem de dados se esse é uma entrada principal ou saída de uma célula. No primeiro caso é retornado o índice da entrada, no segundo os valores l e c, que indicam a linha e a coluna da célula na matriz do circuito. Essa função é utilizada como auxílio pelas funções principais que são responsáveis por calcular o valor de cada saída, partindo das principais e chegando a cada uma das saídas das células. Para isso, obrigatoriamente deve-se passar como argumento um vetor com os valores de entrada. A partir do índice da saída que se deseja, a função chama recursivamente o resultado das saídas que está utilizando como entrada até que se chegue nos valores de entrada. À medida que os valores vão sendo retornados, é executada a função definida na célula e o resultado é passado para quem o chamou. O método responsável por calcular tal resultado nada mais é que um *switch-case* que recebe como entrada o código da função e os argumentos de entrada da célula. Para cada caso é efetuado o cálculo a partir dos valores conforme definido previamentte.

Seguindo mais um passo, a fase de cálculo de resultados recebeu memória para aumentar a performance e diminuir a quantidade de cálculos. Isso se deve ao fato de que a saída de uma mesma célula pode ser usada diversas vezes como entrada de outras. Para essa implementação, a classe recebeu duas novas matrizes de tamanho igual ao número de células, a primeira de valores verdade que indicam se o valor da célula já foi calculado alguma vez e o segundo responsável por registrar o valor calculado. Uma vez que uma célula seja atingida, é verificado o valor verdade. Caso seja positivo, simplesmente é retornado o valor registrado na tabela de resultados. Caso contrário, o valor é calculado normalmente, registrado na tabela de resultados e o flag de validação recebe o valor apropriado. Para quem utiliza a classe, a única diferença é que a partir de então, cada vez que se altere a entrada do circuito é necessário chamar um método que zera todos os flags de validade dos valores das células.

Uma vez que a classe está apta a indicar todos os valores necessários para manipulação dos genótipos, como número de valores inteiros e seus limites, assim como a capacidade de manipular tais valores e gerar resultados a partir de entradas, pode-se implementar a função de fitness para os canditatos. A função deve receber como entrada um genótipo, os dados de tamanho do circuito (*nr* e *nc*), a imagem de referência e a imagem com ruído. Como os cálculos são extremamente demorados, em um primeiro momento a função recebia um vetor de mapeamento, indicando alguns índices de coordenadas de pixels para efetuar o teste com o circuito e fazer a comparação com a

imagem de referência. A implementação da função recebe esse mapa como argumento opcional. Caso seja passado, é assumido que apenas os pixels mapeados serão analisados, senão a imagem inteira é utilizada no cálculo.

8 APRESENTAÇÃO E DISCUSSÃO SOBRE RESULTADOS

O ponto de partida para o presente trabalho foi a implementação do modelo apresentado no artigo base, apenas com algumas modificações. Foram utilizados parâmetros descritos, que são os que seguem:

- a matriz do CGP possui 9 linhas por 7 colunas

- a entrada é composta por uma matriz de 3 x 3 pixels

- a saída é composta por dois valores

- pela característica das funções definidas, cada célula é composta por duas entradas e uma saída. Mesmo as funções que utilizam apenas uma entrada, ou ainda nenhuma, possuem essas conexões para manter a homogneidade do genótipo

- *l* é definido com o valor 1. Essa é a principal modificação e foi feita para efetuar testes também nos valores de restrições de valores.

Na Tabela 8.1 é apresentada a tabela contendo as funções definidas para serem utilizadas pelas células do circuito pelos candidatos. Essas funções são comumente utilizadas em algoritmos não lineares de processamento de imagens. Em todos os testes foram mantidas sem alteração.

Tabela 8.1 – Funções utilizadas no CGP

Código	**Função**	**Descrição**
0	255	constant
1	x	identidade
2	255-x	inversão
3	max(x; y)	máximo
4	min(x; y)	mínimo
5	x >> 1	divisão por 2
6	x >> 2	divisão por 4
7	x + y	adição
8	x +S y	adição com saturação
9	(x + y) >> 1	media
10	y if (x > 127) else x	atribuição condicional
11	|x - y|	diferença absoluta
12	x << 1	multiplicação por 2
13	x << 2	multiplicação por 4

Fote: Criado pelo autor

Todos os testes foram executados utilizando apenas uma imagem como entrada, ela possui 100 pixels de largura por 83 de altura. Essa imagem foi convertida para apenas um canal de cor simplesmente assumindo o canal azul como sendo o de intensidade geral da imagem. Na Figura 8.1 é apresentada a imagem original, na Figura 8.2 a imagem em apenas um canal de cor e na Figura 8.3 um exemplo de imagem com ruído sal-e-pimenta de 10% gerado sinteticamente pelo próprio matlab:

Figura 8.1 – Imagem original

Fonte: Desconhecida

Figura 8.2 – Imagem convertida para um canal de cor

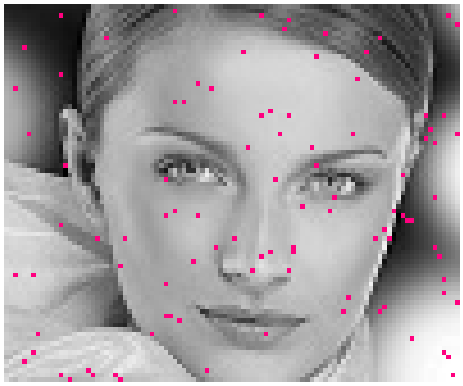
Fonte: Criado pelo autor

Figura 8.3 – Imagem com ruído sal-e-epimenta gerado pelo Matlab

Fonte: Criado pelo autor

O algoritmo foi executado com um mapa de pixels a serem analizados escolhidos aleatoriamente correspondendo a 10% de todos os pixels da imagem. Essas restrições surgiram pelo fato de que a execução da AG para encontrar uma solução é um processo lento. Todo o desenvolvimento foi feito em um PC com processador i3 de 2,27 GHz e 4GB de memória, e um resultado como o demostrado levava cerca de 6 horas para ser encontrado. Na Figura 8.4 podemos ver um exemplo de um desses mapeamentos:

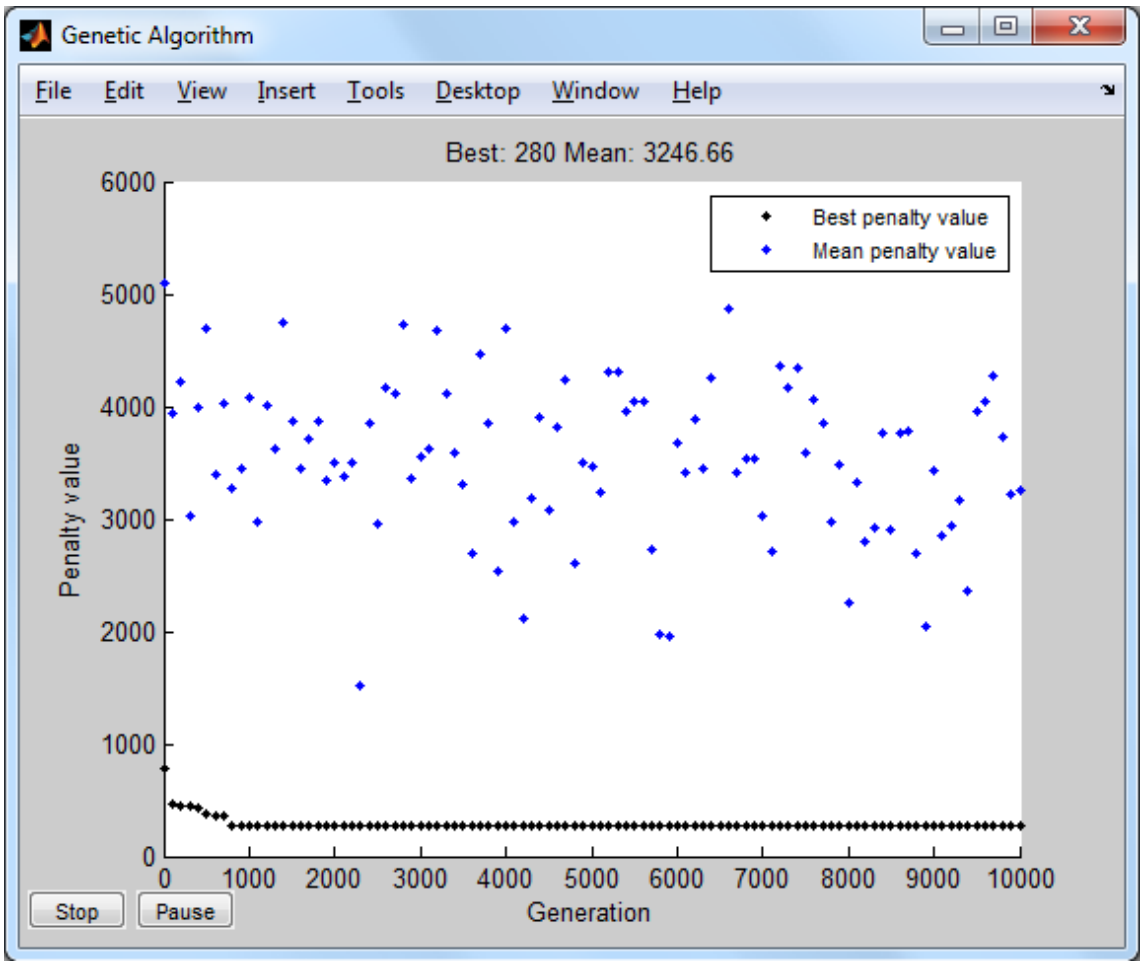
Figura 8.4 – Mapeamento de coordenadas aleatórias



Fonte: Criado pelo autor

A população foi definida como tendo 10 elementos e foram produzidas 10 mil gerações. O resultado do algoritmo genético é apresentado em um gráfico onde são demostrados os valores do melhor valor de penalti e o valor médio de todos os elementos que compõem a população, conforme apresentado na Figura 8.5:

Figura 8.5 – Gráfico gerado pelo Matlab com resultados do Algoritmo Genético



O algoritmo resultante foi aplicado a todos os pixels da imagem com ruído. Na Figura 8.6 podemos ver o resultado da saída que corresponde à indicação de um pixel ser ou não ser ruído. Todas as marcas coloridas representam indicação positiva de pixel defeituoso:

Figura 8.6 – Pixels calculados pelo algoritmo como sendo ruídosos

Fonte: Criado pelo autor

Uma vez que um pixel seja determinado como defeituoso, é utilizada a outra saída para determinar o novo valor que deve ser assumido como verdadeiro, ou seja, o valor filtrado. O resultado final dessa análise e filtragem é apresentado na Figura 8.7:

Figura 8.7 – Resultados da aplicação de filtro

Fonte: Criado pelo autor

O resultado foi bastante animador, uma vez que foram utilizados apenas 10% dos pixels para o levantamento do algoritmo final. Na imagem podemos notar que em partes claras o algoritmo foi mais efetivo em pixels de ruído escuros. Também pode-se notar que alguns pixels foram substituídos por cores muito próximas às cores que deveriam ocupar tais áreas.

Notável também foi o fato de o algoritmo genético já ter encontrado tal solução logo no começo do processo. Alguns fatores devem ter determinado isso. Nos artigos que descrevem e utilizam o CGP, sempre é colocado que uma população pequena é mais eficiente para o processo. Exatamente por esse motivo foi definido dessa forma. Entretanto, os mesmos artigos descrevem uma implementação à parte para os algoritmos de seleção dos indivíduos das novas populações. Nessa implementação, um novo indivíduo gerado, se possuir valores de penalti próximo a um indivíduo que já está na população, tem prioridade para sobreviver. Segundo os autores, isso tira proveito da redundância apresentada pelo CGP no sentido de que seu circuito possui vários caminhos paralelos que podem ir formando um caminho válido com o passar das gerações.

Para contornar esse fato, e testar uma nova possibilidade, foi executado o algoritmo novamente com algumas modificações para que se pudesse gerar uma comparação de resultados:

- as amostras de entrada passam a ser de 5x5 pixels

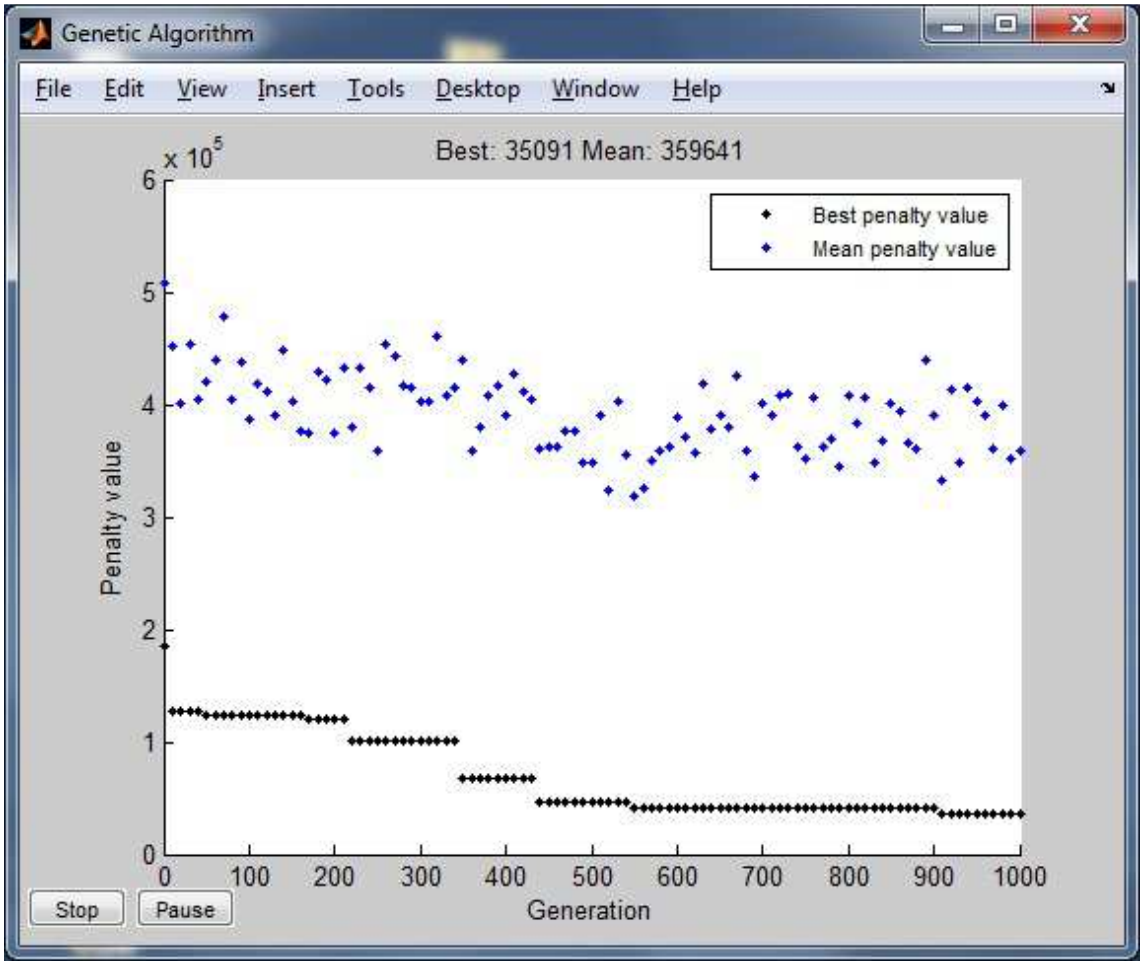
- a população foi definida com 100 indivíduos

- l foi definido como no artigo, com o valor de nc, o que permite conectividade total. Entretanto uma restrição adicional foi imposta, de que somente as primeiras 4 colunas podem ser ligadas às entradas principais

- 1 mil gerações foram criadas

Nessa configuração, o AG levou cerca de 6 dias para apresentar os resultados. Durante a execução o gráfico com dados sobre a população já mostrava dados que pareciam animadores, com o valor de penalidade alcançando melhores marcar ao longo de todo o processo como podemos ver na Figura 8.8:

Figura 8.8 - Gráfico gerado pelo Matlab com resultados do Algoritmo Genético



Fonte: Criado pelo autor

Ao aplicar o algoritmo gerado na imagem ruidosa, pode-se notar claramente uma melhora nos resultados. Dessa fez mais pixels foram detectados como ruído, inclusive aqueles saturados com branco. Notar que os pixels de toda a borda são igonados pelo fato da matriz precisar de 2 vizinhos para o cálculo como já descrito anteriormente.

Figura 8.9 - Resultados da aplicação de filtro

Fonte: Criado pelo autor

Comparação com artigo

Os resultados obtidos não podem ser comparados com os resultados obtidos nos testes do artigo. Em primeiro lugar porque a cada iteração do algoritmo genético foram utilizadas mais de 60 mil imagens com diferentes ruídos enquanto neste trabalho apenas uma imagem com ruído. Além disso, para se chegar na solução comparável com os filtros existentes, foram geradas cerca de 200 mil gerações na obtenção de cada filtro. Para tornar tais testes viáveis, seria necessário alto poder de processamento.

9 CONCLUSÕES E TRABALHOS FUTUROS

O grande objetivo do trabalho foi alcançado, os resultados demonstraram que é uma realidade a utilização de Algoritmos Genéticos para encontrar soluções para a remoção de ruído em imagens. A partir do estudo atual, o algoritmo pode ser amplamente estendido e novas possibilidades podem ser testadas. Não se chegou ao estado de poder utilizar a solução para fins reais mas o conceito foi concretizado e demonstrado, o que possibilita que novos trabalhos podem ser feitos com base no atual. Nessa fase, as alterações e demonstração de resultados exigem muito tempo e processamento, o que torna o processo um pouco lento. Mesmo sendo assim, o estudo é extremamente válido porque a demora está em executar o treinamento do algoritmo, uma vez que se tenha encontrado a solução basta aplicá-la. Outro fato é que, após determinar uma boa base de dados de entrada, o trabalho é inteiramente feito pelo computador, em nenhum momento é mais necessário a intervenção humana. Isso faz com que o processo se torne extremamente barato.

Durante a fase de implementação do trabalho apresentado surgiram algumas ideias que provavelmente poderiam nortear trabalhos futuros sobre o tema e que não foram colocadas em prática por estrapolar o escopo de tempo e ir além dos objetivos iniciais. Entre eles pode-se destacar:

- separar o processo de detecção de ruído e de aplicação de filtro em duas etapas. Ou seja, em primeiro lugar determinar uma função de fitness específica para a primeira etapa e encontrar o melhor circuito para isso. Após, agregar ao circuito a função final para se chegar à solução final utilizando a detecção previamente determinada;

- determinar outras funções de fitness para a fase de filtragem como um todo. Eventualmente funções que penalisem mais a detecção errada de pixels que não são ruído tendam a preservar melhor os detalhes, mesmo sendo menos efetivas na redução geral do ruído;

REFERÊNCIAS

GONZALEZ, Rafael C.; WOODS, Richard E. **Processamento de Imagens Digitais.** 1. ed. São Paulo: Editora Edgard Blücher Ltda, 2000.

GONZALEZ, Rafael C.; WOODS, Richard E.; EDDINS, Steven L. **Digital Image Processing using Matlab**, 1. ed. Upper Saddle River: Pearson Prentice Hall, 2004.

PRATT, William K. **Digital Image Processing**: PIKS Scientific Inside. 4th. ed. Hoboken: Wiley-Interscience, 2007.

VASICEK, Zdenek; BIDLO, Michal. Evolutionary Design of Robust Noise-Specific Image Filters, **IEEE Congress on Evolutionary Computation (CEC)**, New Orleans, LA, p. 269-276, jun. 2011.

SEKAMINA , L.; DRABEK, V. Automatic design of image operators using evolvable hardware. **5th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop**. Brno University of Technology, p. 132–139, 2002.

MILLER, J. F.; THOMSON, P., Cartesian Genetic Programming. **3rd European Conference on Genetic Programming EuroGP2000**, ser. LNCS, vol. 1802. Springer, p. 121–132, 2000.

ANEXO A – CÓDIGO GERADO PARA O MATLAB

Classe rfCandidate

```
% ------------------------------------------------------------------------
% Definição das funções disponíveis para os nodos
% ------------------------------------------------------------------------
% Cod : Função                    Descrissão
%
%   0 : 255                       constant
%   1 : x                         identity
%   2 : 255-x                     inversion
%   3 : max(x; y)                 maximum
%   4 : min(x; y)                 minimum
%   5 : x >> 1                    division by 2
%   6 : x >> 2                    division by 4
%   7 : x + y                     addition
%   8 : x +S y                    addition with saturation
%   9 : (x + y) >> 1              average
%  10 : y if (x > 127) else x     conditional assignment
%  11 : |x - y|                   absolute difference
%  12 : x << 1                    multiplication by 2 with saturation
%  13 : x << 2                    multiplication by 4 with saturation
% ------------------------------------------------------------------------

classdef rfCandidate < handle

    properties
        mTable;
        mPo;
        mResult;
        mValid;
    end

    methods
        % Se 'gen' for passado como argumento, 'po' é ignorado
        % e definido pelo primeiro.
        function obj = rfCandidate(nr, nc, ni, po, gen)
            obj.mTable = zeros(nr,nc,ni+1);
            obj.mResult = zeros(nr,nc);
            obj.mValid = zeros(nr,nc);

            if nargin > 4
                i = 1;
                sif = ni+1;
                for c = 1:nc
                    for r = 1:nr
                      obj.mTable(r,c,:) = gen(i:i+sif-1);
                      i = i+sif;
                    end
                end
                i = nr * nc * sif + 1;
                obj.mPo = gen(i:end);
            else
                obj.mPo = zeros(po,1);
            end
        end
```

```
function gen = Genotype(obj)
    [sr,sc,sif] = size(obj.mTable);
    so = size(obj.mPo);
    gen = zeros(sr * sc * sif + so(1));

    i = 1;
    for c = 1:sc
        for r = 1:sr
          gen(i:i+sif-1) = obj.mTable(r,c,:);
          i = i+sif;
        end
    end
    gen(i:i+so-1) = obj.mPo;
end

%type = 0 : entrada primária de índice 'l'
%type = 1 : saídas de nodo na linha l e coluna c
function [type l c] = outNode(obj, pi, index)
    if index < pi
        type = 0;
        l = index+1;
        c = 0;
    else
        type = 1;
        [nr,~,~] = size(obj.mTable);
        c = floor((double(index)-pi)/nr)+1;
        l = (index - (pi-1)) - ((c-1)*nr);
    end
end

function result = executeFunction(obj, f, nIn)
    switch f
        case 0 % constant
            result = 255;
        case 1 % identity
            result = nIn(1);
        case 2 % inversion
            result = 255 - nIn(1);
        case 3 % maximum
            result = max(nIn(1), nIn(2));
        case 4 % minimum
            result = min(nIn(1), nIn(2));
        case 5 % division by 2
            result = nIn(1) / 2;
        case 6 % division by 4
            result = nIn(1) / 4;
        case 7 % addition
            result = nIn(1) + nIn(2);
        case 8 % addition with saturation
            result = rem(nIn(1) + nIn(2), 256);
        case 9 % average
            result = (nIn(1) + nIn(2))/2;
        case 10 % conditional assignment
            if nIn(1) > 127
                result = nIn(2);
            else
                result = nIn(1);
            end
        case 11 % absolute difference
            result = abs(nIn(1) + nIn(2));
        case 12 % multiplication by 2 with saturation
```

```
            result = rem(nIn(1)*2, 256);
        case 13 % multiplication by 4 with saturation
            result = rem(nIn(1)*4, 256);
        otherwise
            result = 0;
    end
end

function resetCalcState(obj)
    obj.mValid(:) = 0;
end

function result = reduceNode(obj, pIn, l, c)
    if obj.mValid(l,c) == 1
        result = obj.mResult(l,c);
    else
        pi = size(pIn);
        pi = pi(1);

        [~,~,ni] = size(obj.mTable);
        ni = ni - 1;

        nIn = zeros(ni,1);

        for i=1:ni
            index = obj.mTable(l,c,i);

            [nType,nL,nC] = outNode(obj, pi, index);
            if nType == 0
                nIn(i) = double(pIn(nL));
            else
                nIn(i) = reduceNode(obj, pIn, nL, nC);
            end
        end

        result = executeFunction(obj, obj.mTable(l,c,3), nIn);

        obj.mResult(l,c) = result;
        obj.mValid(l,c) = 1;
    end
end

function result = reducePo(obj, pIn, pi, po)
    [type l c] = outNode(obj, pi, obj.mPo(po));
    if type == 0
        result = double(pIn(l));
    else
        result = reduceNode(obj, pIn, l, c);
    end
end

function [lb ub] = getGenotypeBounds(obj, pi)
    [sr,sc,sif] = size(obj.mTable);
    so = size(obj.mPo);
    so = so(1);
    nv = sr * sc * sif + so;

    lb = zeros(nv,1);
    ub = zeros(nv,1);
```

```
            lbc = 0;
            ubc = pi-1;

            i = 1;

            for c = 1:sc
                for r = 1:sr
                    for ni = 0:sif-2
                        % Os índices das entradas devem ser as saídas
da
                        % coluna anterior (ou as pi´s).
                        lb(i+ni) = lbc;
                        ub(i+ni) = ubc;
                    end

                    lb(i+sif-1) = 0;
                    ub(i+sif-1) = 13;

                    i = i+sif;
                end

                lbc = ubc + 1;
                ubc = lbc + sr -1;
            end

            lb(i:end) = lbc;
            ub(i:end) = ubc;
        end

        function nv = getGenotypeSize(obj)
            [sr,sc,sif] = size(obj.mTable);
            so = size(obj.mPo);
            nv = sr * sc * sif + so(1);
        end

    end % methods

end % classdef rfCandidate
```

Função de fitness com amostras de 5x5 pixels

```
function fit = rfCandidateFitness5x5(gen, nr, nc, Ic, In, map)

    obj = rfCandidate(nr, nc, 2, 0, gen);

    fit = 0;

    if nargin > 5
        [sm,~] = size(map);
        for i = 1:sm
            y = map(i,1);
            x = map(i,2);
            sample = In(y-2:y+2, x-2:x+2);

            mat    =    [sample(1:5),    sample(6:10),    sample(11:15),
sample(16:20), sample(21:25)];

            s = obj.reducePo(mat', 25, 1);
            if s > 127
                fit  =  fit  +  abs(double(Ic(y,x))  -  obj.reducePo(mat',
25, 2));
            else
                fit = fit + abs(double(Ic(y,x)) - double(In(y,x)));
            end

            obj.resetCalcState();
        end
    else
        [sy,sx] = size(In);

        for i = 3:sy-2
            for j = 3:sx-2
                sample = In(i-2:i+2, j-2:j+2);

                mat   =   [sample(1:5),   sample(6:10),   sample(11:15),
sample(16:20), sample(21:25)];

                s = obj.reducePo(mat', 25, 1);
                if s > 127
                    fit     =     fit     +     abs(double(Ic(i,j))     -
obj.reducePo(mat', 25, 2));
                else
                    fit     =     fit     +     abs(double(Ic(i,j))     -
double(In(i,j)));
                end

                obj.resetCalcState();
            end
        end
    end
end
```

Função de aplicação do resultado na imagem de testes com amostras de 5x5 pixels

```
function rfExecute(gen)

    %-------------------------------------------------------------

    Ir = imread('rosto.jpg');
    Ic = mat2gray(Ir(:,:,2));
    Ic = im2uint8(Ic);
    In = imnoise(Ic, 'salt & pepper', .1);

    %-------------------------------------------------------------

    obj = rfCandidate(9,7,2,2,gen);

    %-------------------------------------------------------------

    [sy,sx] = size(In);

    Io = zeros(sy,sx);

    for i = 3:sy-2
        for j = 3:sx-2
            sample = In(i-2:i+2, j-2:j+2);

            mat    =    [sample(1:5),    sample(6:10),    sample(11:15),
sample(16:20), sample(21:25)];

            s = obj.reducePo(mat', 25, 1);
            if s > 127
                Io(i,j) = obj.reducePo(mat', 25, 2);
            else
                Io(i,j) = In(i,j);
            end

            obj.resetCalcState();
        end
    end

    imwrite(In, 'in.tif');
    imwrite(Io, 'io.tif');

end
```

Função de testes com amostras de 5x5 pixels

```
function s = rfTest5x5Full

    %-------------------------------------------------------------

    Ir = imread('rosto.jpg');
    Ic = mat2gray(Ir(:,:,2));
    Ic = im2uint8(Ic);
    In = imnoise(Ic, 'salt & pepper', .2);

    rfCandidateFitness5x5Args = @(gen)rfCandidateFitness5x5(gen, 9, 7,
Ic, In);
    options = gaoptimset('StallGenLimit', 1000,'Generations', 1000,
'PopulationSize', 100, 'PlotInterval', 10, 'PlotFcns', @gaplotbestf);

    %-------------------------------------------------------------

    obj = rfCandidate(9,7,2,2);
    nv = obj.getGenotypeSize()
    [lb,ub] = obj.getGenotypeBounds(25,7,4);

    %-------------------------------------------------------------

    intCon = 1;
    for i = 2:nv
        intCon = intCon:i;
    end

    %-------------------------------------------------------------

    [s,r,flag] = ga(rfCandidateFitness5x5Args, nv, [], [], [], [], lb,
ub, [], intCon, options)

end
```