DANIEL POPP COELHO

Implementação paralela e automaticamente configurável de
segmentação em imagens pelo algoritmo de Level Sets

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do título de Mestre em
Ciências

São Paulo

2020

DANIEL POPP COELHO

Implementação paralela e automaticamente configurável de
segmentação em imagens pelo algoritmo de Level Sets

Versão Corrigida

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do título de Mestre em
Ciências

Programa: Engenharia Elétrica

Área de concentração: Engenharia
Biomédica

Orientador: Prof. Dr. Sérgio Shiguemi Furuie

São Paulo

2020

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 23 de Setembro de 2020

Assinatura do autor: Daniel P. Coelho

Assinatura do orientador: Sergio Furui

Catálogo-na-publicação

Coelho, Daniel

Implementação paralela e automaticamente configurável de segmentação em imagens pelo algoritmo de Level Sets / D. Coelho -- versão corr. -- São Paulo, 2020.

157 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Telecomunicações e Controle.

1.PROCESSAMENTO DIGITAL DE IMAGENS I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Telecomunicações e Controle II.t.

FOLHA DE AVALIAÇÃO

Nome: COELHO, Daniel Popp

Título: Implementação paralela e automaticamente configurável de segmentação em imagens pelo algoritmo de Level Sets

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do título de Mestre em
Ciências

Aprovado em: _____

Banca examinadora

Prof. Dr. _____

Instituição: _____

Julgamento: _____

Prof. Dr. _____

Instituição: _____

Julgamento: _____

Prof. Dr. _____

Instituição: _____

Julgamento: _____

AGRADECIMENTOS

Agradeço em primeiro lugar ao meu orientador Prof. Dr. Sérgio Shiguemi Furuie, pela orientação, paciência e atenção.

Agradeço à minha esposa e minha família por todo o carinho e suporte.

Agradeço aos professores Dr. Hae Yong Kim e Dr. John Sims, que gentilmente aceitaram o convite para minha banca de qualificação, levantando importantes pontos que ajudaram a direcionar o meu trabalho.

Os resultados publicados aqui são em parte baseados nos dados gerados pela TCGA *Research Network*: <http://cancergenome.nih.gov/>.

Dados usados nessa publicação foram gerados pelo *National Cancer Institute Clinical Proteomic Tumor Analysis Consortium (CPTAC)*.

RESUMO

A segmentação de imagens é o processo de identificar e delinear objetos contidos nestas. É uma operação crucial em diversos ramos de computação baseada em imagens, como visualização 3D, filtragem, interpolação e registro. Técnicas semiautomáticas segmentam objetos programaticamente a partir de uma entrada inicial providenciada pelo usuário, sendo o algoritmo *Distance-Regularized Level-Set Evolution*, ou DRLSE (LI, 2010a), uma destas técnicas. A determinação dos valores de parâmetros de algoritmos como este não é trivial, sendo necessário que o usuário possua conhecimento profundo da implementação e teoria por trás do algoritmo. Nesse contexto são utilizados sistemas que automatizam a tarefa de escolha de parâmetros, denominados configuradores de algoritmos. Os algoritmos genéticos são métodos de busca ou otimização baseado em mecanismos de seleção natural, e podem ser usados para a otimização dos parâmetros de um algoritmo como o DRLSE de modo robusto e generalizado. Uma das desvantagens de tais algoritmos é a necessidade de avaliar o algoritmo alvo diversas vezes, que pode ser custoso. Felizmente, o algoritmo alvo DRLSE é altamente paralelizável. Esse trabalho visa utilizar a plataforma CUDA para aceleração do algoritmo DRLSE em GPU, permitindo sua rápida e repetida execução por um algoritmo genético, por sua vez permitindo a determinação dos parâmetros ideais com base apenas em um exemplo inicial e uma segmentação de referência. A implementação em CUDA do algoritmo DRLSE desenvolvida nesse trabalho apresenta ganho de performance de 500 vezes em relação à implementação de referência em MATLAB, completando 1000 iterações em menos de um segundo para imagens de 1024x1024 pixels. O sistema final desenvolvido, apesar de apresentar certas limitações, é robusto o suficiente para lidar com contextos de segmentação diferentes, e permite a configuração automática dos parâmetros do algoritmo DRLSE em regime semi-interativo, e aplicação do algoritmo DRLSE, com os parâmetros determinados automaticamente, em regime interativo.

Palavras-chave: DRLSE. Level Sets. Segmentação. CUDA. Algoritmos Genéticos.

ABSTRACT

Image segmentation is the process of identifying and delineating objects contained in them. It is a crucial operation in many different fields of image processing, like 3D visualization, filtering, interpolation and registration. Semiautomatic techniques segment images programmatically, based on user input, with the *Distance-Regularized Level-Set Evolution* or DRLSE (LI, 2010a) being one of those techniques. Choosing the correct values for the parameters of algorithms is not a trivial task, demanding that the user have deep knowledge of the implementation and theory behind the algorithm. Systems that automate the task of choosing those parameters are used in this context, and are named algorithm configurators. Genetic algorithms are search and optimization methods based on natural selection mechanisms, that can also be used as algorithm configurators to optimize the parameters of an algorithm like DRLSE in a robust and generalized way. One of the disadvantages of such algorithms is the need to repeatedly evaluate the target algorithm, which can be costly. Fortunately, the DRLSE algorithm is highly parallelizable. This work aims to use the CUDA platform for acceleration of the DRLSE algorithm in a GPU, allowing its fast and repeated execution by a genetic algorithm, in turn allowing the determination of the ideal parameters based solely on an initial sample and reference segmentation. The CUDA implementation of the DRLSE algorithm developed here shows a 500 times performance gain with respect to the reference MATLAB implementation, achieving 1000 iterations in less than a second for 1024x1024 pixel images. The final system, despite of some limitations, is robust enough to handle different segmentation situations, allowing the automatic configuration of the DRLSE algorithm's parameters to be performed in a semi-interactive regime, as well as the application of the DRLSE algorithm, with the automatically determined parameters, in an interactive regime.

Keywords: DRLSE. Level Sets. Segmentation. CUDA. Genetic Algorithms.

LISTA DE FIGURAS

Figura 1 – Esquerda: Imagem 73, série 701-sT1W3DTFEAX PRE-30908, estudo 04-23-2000-MR BRAIN WOW CONTRAST-18837, paciente C3L-00629. Fonte: CPTAC, 2018a. Originalmente 528x528 pixels. Modalidade MR. Direita: Mesma imagem com uma segmentação binária.....	35
Figura 2 – Fluxograma de um algoritmo genético. Fonte: Daniel Coelho.....	46
Figura 3 – Visualização do processo de seleção proporcional. Fonte: Daniel Coelho.....	51
Figura 4 – Visualização do processo de seleção por amostragem universal estocástica. Fonte: Daniel Coelho.....	52
Figura 5 – Diferentes metodologias de recombinação. Os cromossomos originais (acima) são recombinados, originando novos cromossomos (abaixo). Fonte: Daniel Coelho.....	53
Figura 6 – Espaço de parâmetros varrido pelo operador de recombinação BLX- α aplicado à dois indivíduos a e b . Fonte: Daniel Coelho.....	55
Figura 7 – Espaço de parâmetros com cromossomos binários de 3 caracteres. Em azul, o hiperplano descrito pelo esquema $H2 = (0,*,*)$. Fonte: Daniel Coelho.....	59
Figura 8 – Exemplo de processamento sequencial de uma matriz bidimensional $I[lin][col]$, com operador $proc(k)$ aplicado a cada pixel. Fonte: Daniel Coelho.....	64
Figura 9 – Exemplo de processamento paralelo por meio de um <i>kernel processPixel</i> . Uma matriz bidimensional $I[lin][col]$ tem cada pixel processado com um operador $proc(k)$. Fonte: Daniel Coelho.....	65
Figura 10 – Comparação entre abstração em <i>software</i> nas estruturas <i>thread</i> , bloco e <i>grid</i> com a implementação em <i>hardware</i> das estruturas <i>GPU</i> , SM e núcleo. Fonte: Daniel Coelho.....	66
Figura 11 – Esquematização de uma placa de vídeo, indicando os diversos tipos de memória e caches usados. Fonte: Daniel Coelho.....	68

Figura 12 – Imagem 116, série 7-Ax 3D DualEcho BH-37647, estudo 03-24-2003-MR ABDOMEN NONENHANCED ENHANCEDAB-13326, paciente C3L-00598. Fonte: CPTAC, 2018b. Originalmente 512x512 pixels. Modalidade MR. Imagem superior: sem segmentações. Imagem inferior: com contorno de referência em verde, e contorno inicial em vermelho.....	73
Figura 13 – Imagem 113, estudo MiMM_SBILab. Fonte: GUPTA, 2019. Originalmente 2560x1920 pixels. Imagem de microscopia. À esquerda sem segmentações, à direita com contornos de referência em verde, e contorno inicial em vermelho.....	74
Figura 14 – Imagem 77, série 801-sT1W3DTFEAX POST-99314, estudo 04-23-2000-MR BRAIN WOW CONTRAST-18837, paciente C3L-00629. Fonte: CPTAC, 2018a. Originalmente 512x512 pixels. Modalidade MR. À esquerda sem segmentações, à direita com contorno de referência em verde, e contorno inicial em vermelho.....	75
Figura 15 – Imagem 73, série 701-sT1W3DTFEAX PRE-30908, estudo 04-23-2000-MR BRAIN WOW CONTRAST-18837, paciente C3L-00629. Fonte: CPTAC, 2018a. Originalmente 528x528 pixels. Modalidade MR. À esquerda sem segmentações, à direita com contorno de referência em verde, e contorno inicial em vermelho.....	76
Figura 16 – Imagem 588, série abdominallymphnodes-03108, estudo 09-14-2014-ABDLYMPH005-abdominallymphnodes-90939, paciente ABD_LYMPH_005. Fonte: ROTH, 2015. Originalmente 512x512 pixels. Modalidade CT. Acima sem segmentações, abaixo com contornos de referência em verde, e contorno inicial em vermelho.....	77
Figura 17 – Esquematização do sistema desenvolvido. Fonte: Daniel Coelho.....	78
Figura 18 – Interface gráfica desenvolvida, exibindo a imagem descrita na Fig. 13, em processo de delimitação de um contorno, em verde. Fonte: Daniel Coelho.....	80
Figura 19 – Diferentes opções para o uso do algoritmo DRLSE. A: Contornos iniciais, e de referência (<i>gold-standard</i>) são usados para treinamento, originando um conjunto de parâmetros. B: Um conjunto de parâmetros e um contorno inicial são usados para uma segmentação simples. Fonte: Daniel Coelho.....	80

Figura 20 – Rasterização de um contorno poligonal (vermelho) em uma segmentação binária, onde pixels em preto descrevem áreas fora do contorno, e pixels em cinza descrevem áreas dentro do contorno. Fonte: Daniel Coelho.....	81
Figura 21 – Progresso no processamento de uma seção transversal de uma segmentação. À esquerda: Contorno rasterizado. No centro: Contorno rasterizado mapeado ao intervalo $[-c_0, c_0]$ (ϕ inicial). À direita: ϕ após algumas iterações. Fonte: Daniel Coelho.....	82
Figura 22 – Imagem em preto e branco (à esquerda), e sua transformada de distância (à direita), indicando a mínima distância ao pixel branco mais próximo. Valores baixos de distância são representados com tons escuros de cinza. Fonte: Daniel Coelho....	94
Figura 23 – Cenários para comparação das métricas de avaliação de segmentação. Em cada um, de A a J, a cor verde indica o contorno de referência <i>gold-standard</i> , e a cor vermelha indica um outro contorno qualquer. Fonte: Daniel Coelho.....	98
Figura 24 – A: Imagem DICOM utilizada para validação. Fonte: ERICKSON, 2016; B: Região segmentada ampliada. O quadrado branco indica o contorno inicial, e o círculo lilás indica a região ampliada em C. Tracejado vermelho: Conjunto nível zero obtido da implementação em MATLAB após 1000 iterações; Linha contínua verde: Análogo para a implementação em CUDA; Linha pontilhada em azul: Análogo pra implementação em C++ serial.....	100
Figura 25 – Análise de operadores de seleção para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, Q_S médio é exibido, e as 20 últimas gerações são ampliadas à direita.....	108
Figura 26 – Análise de operadores de seleção para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, o desvio padrão de Q_S é exibido, e as 20 últimas gerações são ampliadas à direita.....	109
Figura 27 – Análise de operadores de recombinação (grupo 1) com diferentes valores de P_C para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, Q_S médio é exibido, e as 20 últimas gerações são ampliadas à direita.....	110

Figura 28 – Análise de operadores de recombinação (grupo 1) com diferentes valores de P_C para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, o desvio padrão de Q_S é exibido, e as 20 últimas gerações são ampliadas à direita. Operadores de recombinação dos demais grupos são exibidos em cinza....111

Figura 29 – Análise de operadores de recombinação (grupo 2) com diferentes valores de P_C para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, Q_S médio é exibido, e as 20 últimas gerações são ampliadas à direita. Operadores de recombinação dos demais grupos são exibidos em cinza.....112

Figura 30 – Análise de operadores de recombinação (grupo 2) com diferentes valores de P_C para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, o desvio padrão de Q_S é exibido, e as 20 últimas gerações são ampliadas à direita. Operadores de recombinação dos demais grupos são exibidos em cinza....113

Figura 31 – Análise de operadores de recombinação (grupo 3) com diferentes valores de P_C para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, Q_S médio é exibido, e as 20 últimas gerações são ampliadas à direita. Operadores de recombinação dos demais grupos são exibidos em cinza.....114

Figura 32 – Análise de operadores de recombinação (grupo 3) com diferentes valores de P_C para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, o desvio padrão de Q_S é exibido, e as 20 últimas gerações são ampliadas à direita. Operadores de recombinação dos demais grupos são exibidos em cinza....115

Figura 33 – Análise de operadores de mutação com diferentes valores de P_M para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, Q_S médio é exibido, e as 20 últimas gerações são ampliadas à direita.....116

Figura 34 – Análise de operadores de mutação com diferentes valores de P_M para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, o desvio padrão de Q_S é exibido, e as 20 últimas gerações são ampliadas à direita.....117

Figura 35 – Varredura do espaço de parâmetros para o contexto de segmentação ‘abdome’. Cada pixel descreve o valor de *fitness* obtido da segmentação usando os

correspondentes valores de α , λ , e os demais valores indicados na Tab. 11. O ponto em vermelho indica os valores de α e λ mostrados na Tab. 11.....120

Figura 36 – Varredura do espaço de parâmetros para o contexto de segmentação ‘celulas1’. Cada pixel descreve o valor de *fitness* obtido da segmentação usando os correspondentes valores de α , λ , e os demais valores indicados na Tab. 11. O ponto em vermelho indica os valores de α e λ mostrados na Tab. 11.....121

Figura 37 – Varredura do espaço de parâmetros para o contexto de segmentação ‘cerebro1’. Cada pixel descreve o valor de *fitness* obtido da segmentação usando os correspondentes valores de α , λ , e os demais valores indicados na Tab. 11. O ponto em vermelho indica os valores de α e λ mostrados na Tab. 11.....121

Figura 38 – Varredura do espaço de parâmetros para o contexto de segmentação ‘cerebro2’. Cada pixel descreve o valor de *fitness* obtido da segmentação usando os correspondentes valores de α , λ , e os demais valores indicados na Tab. 11. O ponto em vermelho indica os valores de α e λ mostrados na Tab. 11.....122

Figura 39 – Varredura do espaço de parâmetros para o contexto de segmentação ‘pelvis’. Cada pixel descreve o valor de *fitness* obtido da segmentação usando os correspondentes valores de α , λ , e os demais valores indicados na Tab. 11. O ponto em vermelho indica os valores de α e λ mostrados na Tab. 11.....122

Figura 40 – Resultado da segmentação com parâmetros obtidos pelo processo de treinamento. Vermelho: Contorno inicial; Azul: Contorno final após segmentação com algoritmo DRLSE; Verde: Contorno *gold-standard*. Acima: Contexto ‘abdome’ e respectiva vista ampliada. A baixo: Contexto ‘células1’ e respectiva vista ampliada.....124

Figura 41 – Resultado da segmentação com parâmetros obtidos pelo processo de treinamento. Vermelho: Contorno inicial; Azul: Contorno final após segmentação com algoritmo DRLSE; Verde: Contorno *gold-standard*. Acima: Contexto ‘cérebro1’ e respectiva vista ampliada. Abaixo: Contexto ‘cérebro2’ e respectiva vista ampliada.....125

Figura 42 – Resultado da segmentação com parâmetros obtidos pelo processo de treinamento para o contexto de segmentação ‘pelvis1’. Vermelho: Contorno inicial; Azul: Contorno final após segmentação com algoritmo DRLSE; Verde: Contorno *gold-standard*. A direita: Vista ampliada.....126

Figura 43 – Contextos de segmentação com *phantom*. A: Sem ruído adicional. Em vermelho: Contorno inicial; Em verde: Contorno *gold-standard*. B: *phantom* adicionado de ruído normal com desvio padrão de valor 250. Fonte: Daniel Coelho.....127

Figura 44 – A: Resultado do treinamento com *phantom* sem ruído. Em vermelho: Contorno inicial; Em azul: Contorno final. B: Vista ampliada do canto superior-direito de A. Fonte: Daniel Coelho.....128

Figura 45 – Teste de sensibilidade ao contorno inicial no contexto de segmentação ‘pelvis1’. Vermelho: Contorno inicial; Azul: Contorno final após segmentação com algoritmo DRLSE; Verde: Contorno *gold-standard*. A: Situação de treinamento; B e C: Contornos iniciais alternativos e respectivos contornos finais. A seta vermelha indica uma borda adicional envolvida pelo contorno inicial no caso B.....132

Figura 46 – Teste de sensibilidade ao contorno inicial no contexto de segmentação ‘células1’. Vermelho: Contorno inicial; Azul: Contorno final após segmentação com algoritmo DRLSE; Verde: Contorno *gold-standard*. A: Situação de treinamento; B e C: Contornos iniciais alternativos e respectivos contornos finais.....133

Figura 47 – Testes usando o conjunto de parâmetros treinados no contexto de segmentação ‘celulas1’, indicado na Fig. 13. Vermelho: Contorno inicial; Azul: Contorno final. Acima: Imagem 415 e vista ampliada, estudo MiMM_SBILab. Abaixo: Imagem 506 e vista ampliada, estudo MiMM_SBILab. Fonte: GUPTA, 2019.....135

Figura 48 – Testes usando o conjunto de parâmetros treinados no contexto de segmentação ‘cerebro1’, indicado na Fig. 14. Vermelho: Contorno inicial; Azul: Contorno final. Imagem 165 e vista ampliada, série 801-sT1W3DTFEAX POST-99314, estudo 04-23-2000-MR BRAIN WOW CONTRAST-18837, paciente C3L-00629. Fonte: CPTAC, 2018a.....136

Figura 49 – Testes usando o conjunto de parâmetros treinados no contexto de segmentação ‘pelvis1’, indicado na Fig. 16. Vermelho: Contorno inicial; Azul: Contorno final. Acima: Imagem 269 e vista ampliada, série abdominallymphnodes-03108, paciente ABD_LYMPH_005; Centro: Imagem 183 e vista ampliada, série abdominallymphnodes-51545, paciente ABD_LYMPH_006. Abaixo: Imagem 206 e vista ampliada, série abdominallymphnodes-03108, paciente ABD_LYMPH_005. Fonte: ROTH, 2015.....137

Figura 50 – Captura de tela do programa NVVP durante a análise da execução de uma segmentação com o programa desenvolvido em CUDA. A: Intervalo de 1000 iterações de uma segmentação com algoritmo DRLSE. B: Intervalo entre duas segmentações em CUDA.....140

LISTA DE TABELAS

Tabela 1 – Exemplos de codificação.....	45
Tabela 2 – Seleção por ranking linear.....	49
Tabela 3 – Seleção por ranking exponencial.....	50
Tabela 4 – Contextos de segmentação analisados.....	72
Tabela 5 – Parâmetros controlados pelo usuário.....	88
Tabela 6 – Operadores genéticos implementados.....	89
Tabela 7 – Comparação das métricas de avaliação de segmentação.....	97
Tabela 8 – <i>Occupancy</i> e performance como uma função da restrição de registros.....	104
Tabela 9 – Operadores genéticos usados como padrão.....	106
Tabela 10 – Metaparâmetros finais para o algoritmo genético.....	119
Tabela 11 – Conjunto de parâmetros com melhor <i>fitness</i> por contexto de segmentação.....	120
Tabela 12 – Médias e desvios-padrões de Q_s para 10 treinamentos com <i>phantom</i>	128
Tabela 13 – Médias e desvios-padrões de 10 treinamentos para cada contexto.....	129
Tabela 14 – Comparação da performance para uma segmentação individual.....	139
Tabela 15 – Tempo de execução de um treinamento com 30 indivíduos e 60 gerações.....	141

LISTA DE ALGORITMOS

Algoritmo 1 – Imagem indicadora de bordas $g(x, y)$, CPU.....	83
Algoritmo 2 – <i>EdgeIndicatorKernel</i> , GPU.....	84
Algoritmo 3 – Iteração de DRLSE, CPU.....	84
Algoritmo 4 – <i>LevelSetKernel</i> , GPU.....	85
Algoritmo 5 – <i>DistRegPre</i> , GPU.....	86
Algoritmo 6 – <i>DiracDelta</i> , GPU.....	86
Algoritmo 7 – <i>CoefDICE</i> , CPU.....	91
Algoritmo 8 – <i>CoefDICEKernel</i> , GPU.....	92
Algoritmo 9 – Distância de Hausdorff, CPU.....	94
Algoritmo 10 – <i>ImageOfMins</i> , GPU.....	95

LISTA DE ABREVIATURAS E SIGLAS

AGCR – Algoritmo genético de codificação real

API – *Application Programming Interface* – Interface de Programação de Aplicativos

BLX- α – Operador de recombinação desenvolvido por Eschelman e Shaffer, 1993

cache L1 – Memória de maior capacidade e menor velocidade que registros

cache L2 – Memória de maior capacidade e menor velocidade que cache L1

CPU – *Central Processing Unit* – Unidade de Processamento Central

CT – *Computed Tomography* – Tomografia Computadorizada

CUDA – *Compute Unified Device Architecture* – Arquitetura unificada de dispositivo para computação

DICOM – *Digital Imaging and Communications in Medicine* – Imageamento e Comunicação Digital na Medicina

DRLSE – *Distance-Regularized Level Set Evolution* – Evolução de *Level Set* Regularizada por Distância

FLOPS – *Floating Point Operations Per Second* – Operações de Ponto Flutuante Por Segundo, métrica de performance para um processador

GAC – *Geodesic Active Contours* – Contornos ativos geodésicos, técnica de segmentação

GPU – *Graphics Processing Unit* – Unidade de Processamento de Gráficos, tipo de processador com vários núcleos, e especializado em computação paralela

GPGPU – *General-Purpose Graphics Processing Unit* – Unidade de Processamento de Gráficos de Propósito Genérico, abordagem de usar programas não-relacionados à gráficos em GPUs

HD – *Hausdorff Distance* – Distância de Hausdorff, métrica para comparação entre dois conjuntos

HDa – Distância de Hausdorff adaptada, métrica ajustada para comparação entre dois conjuntos

MR – *Magnetic Resonance* – Ressonância Magnética, técnica de imageamento médico

NVVP – *NVIDIA Visual Profiler* – Analisador de Performance Visual da NVIDIA, ferramenta usada para análise em performance de *kernels*

SBX – *Simulated Binary Crossover* – Recombinação Binária Simulada, operador de recombinação usado em um algoritmo genético

SIMD – *Single Instruction Multiple Data* – Instrução Única Múltiplos Dados, técnica de processamento paralelo

SM – *Streaming Multiprocessors* – Multiprocessador de *Streaming*, agrupamento de núcleos em uma GPU

spel – *Spatial Element* – Elemento Espacial, correspondendo à um *pixel* ou um *voxel* dependendo do contexto

SUS – *Stochastic Universal Sampling* – Amostragem Universal Estocástica, um operador genético de seleção

TFLOPS – *Trillion Floating Point Operations Per Second* – Trilhões de Operações de Ponto Flutuante Por Segundo, métrica de performance para um processador

VLIW – *Very Long Instruction Word* – Palavra de Instrução Muito Longa, técnica de programação paralela

VTK – *Visualization Toolkit* – Kit de Ferramentas de Visualização, biblioteca para visualização de dados

ZLS – *Zero-Level Set* – Conjunto Nível Zero, conjunto de *spels* que apresentam zero para o valor de ϕ

LISTA DE SÍMBOLOS

$\vec{r}(c)$ – Coordenadas de um *spel* c

$v(c)$ – Valor de um *spel* c

$I(x, y)$ – Imagem de entrada a segmentar

S_k^m – Partição k de uma imagem I em m diferentes subconjuntos, $\{S_k^1, S_k^2, \dots, S_k^m\}$

B_{S_k} – Imagem binária gerada a partir da segmentação S_k

ϕ – Função de *Level Set*, principal função definida na formulação de um algoritmo de *Level Sets*, como o DRLSE

$\mathcal{E}(\phi)$ – Energia total de ϕ , minimizada no algoritmo DRLSE

μ – Peso do termo de regularização na formulação do algoritmo DRLSE, usado no modo inicial da implementação

$\mathcal{R}_p(\phi)$ – Termo de regularização na formulação da energia total de ϕ , no algoritmo DRLSE

$\mathcal{E}_{ext}(\phi)$ – Energia externa de ϕ na formulação do algoritmo DRLSE

$p(s)$ – Energia potencial na formulação do algoritmo DRLSE

λ – Peso do termo de comprimento na formulação do algoritmo DRLSE, usado no modo inicial da implementação

$g(x, y)$ – Imagem indicadora de bordas, usada no algoritmo DRLSE

$\delta_\varepsilon(s)$ – Aproximação da função Delta de Dirac

ε – Parâmetro usado nas aproximações da função de Heaviside e Delta de Dirac, parte da formulação do algoritmo DRLSE

α – Peso do termo de área na formulação do algoritmo DRLSE, usado no modo inicial da implementação

H_ε – Aproximação da função de Heaviside

τ – *Timestep* – Intervalo de tempo, intervalo entre iterações na formulação do algoritmo DRLSE, usado no modo inicial da implementação

G_σ – Núcleo gaussiano com desvio padrão σ

σ – Desvio padrão do núcleo gaussiano usado na geração da imagem de bordas

DICE – Coeficiente de Dice, métrica de similaridade entre dois conjuntos

HD, $HD(A, B)$ – *Hausdorff Distance* – Distância de Hausdorff, métrica de avaliação de similaridade entre dois conjuntos A e B

$h(A, B)$ – Distância de Hausdorff direcionada entre conjuntos A e B

N – População de um algoritmo genético

f_i – *fitness* de um indivíduo i

P_i – Probabilidade de seleção de um certo indivíduo i

M – *Mating Pool* – População Intermediária, conjunto de indivíduos da população que passaram pelo processo de seleção

T_o – Tamanho do torneio no operador de seleção por torneio

r_i – Ranking de um indivíduo i nos operadores de seleção por ranking linear e exponencial

η^- – Probabilidade de seleção do indivíduo de pior *fitness*, no método de seleção por ranking linear

η^+ – Probabilidade de seleção do indivíduo de melhor *fitness*, no método de seleção por ranking linear

b – Parâmetro base no operador de seleção por ranking exponencial

e_i – Número de descendentes de um indivíduo i esperados para uma geração

d_i – Número de descendentes de um indivíduo i observados em uma geração

rand – Valor de ponto flutuante aleatório contido no intervalo $[0,1]$, proveniente de distribuição uniforme

T_r – Valor limítrofe na seleção por truncamento

P_c – Probabilidade de aplicação do operador de recombinação em uma geração de um algoritmo genético

α_{BLX} – Parâmetro do operador de recombinação BLX- α

β_i – Fator de espalhamento, parâmetro usado no operador de recombinação SBX

η_{SBX} – Índice de distribuição, parâmetro que descreve o formato de β_i na formulação do operador de recombinação SBX

P_M – Probabilidade de aplicação do operador de mutação em uma geração de um algoritmo genético

σ_m – Desvio padrão usado no operador de mutação gaussiana

H – Esquema na formulação da teoria dos esquemas

n – Número de caracteres por cromossomo, na formulação de um algoritmo genético

$Ord(H)$ – Ordem de um esquema H na teoria dos esquemas

$\Delta(H)$ – Comprimento definatório de um esquema H

$f(H, t)$ – *fitness* de um esquema H em uma geração t

t – Uma geração de um algoritmo genético na formulação da teoria dos esquemas

$\xi(H, t)$ – Representatividade de um esquema H em uma geração t , na teoria dos esquemas

$\overline{f(t)}$ – *fitness* médio de todos os cromossomos na geração t

c_0 – Constante na formulação do algoritmo DRLSE; Amplitude da imagem binária inicial

i_l – *late iter* – Iteração tardia, fração das iterações da implementação do algoritmo em DRLSE executada em modo inicial

μ_l – Peso do termo de regularização na formulação do algoritmo DRLSE, usado no modo tardio da implementação

λ_l – Peso do termo de comprimento na formulação do algoritmo DRLSE, usado no modo tardio da implementação

α_l – Peso do termo de área na formulação do algoritmo DRLSE, usado no modo tardio da implementação

τ_l – *Timestep* – Intervalo de tempo, intervalo entre iterações na formulação do algoritmo DRLSE, usado no modo tardio da implementação

T – Número máximo de iterações permitidas para o algoritmo DRLSE

GS – *Gold-Standard*, Padrão-ouro, imagem ou dados de referência

HDa – Distância de Hausdorff adaptada

Q_s – Métrica combinada, avalia a similaridade entre dois conjuntos combinando Distância de Hausdorff e coeficiente de Dice

SUMÁRIO

1. Introdução	25
2. Objetivos	30
3. Fundamentação teórica	31
3.1. Imagens Médicas	31
3.2. Segmentação de imagens	34
3.2.1. <i>Level Sets</i>	35
3.3. Avaliação de algoritmos e resultados de segmentação	41
3.3.1. Coeficiente de Dice	42
3.3.2. Distância de Hausdorff	43
3.4. Algoritmos genéticos	43
3.4.1. Inicialização	45
3.4.2. Avaliação de <i>fitness</i>	45
3.4.3. Seleção	47
3.4.3.1. Seleção por torneios	48
3.4.3.2. Seleção por ranking linear	49
3.4.3.3. Seleção por ranking exponencial	50
3.4.3.4. Amostragem universal estocástica	50
3.4.3.5. Seleção por truncamento	52
3.4.4. Recombinação	52
3.4.4.1. Recombinação aritmética	54
3.4.4.2. Recombinação BLX- α	54
3.4.4.3. Recombinação binária simulada	55
3.4.5. Mutação	57

3.4.5.1.	Mutação uniforme	57
3.4.5.2.	Mutação gaussiana	57
3.4.6.1.	Teoria de Esquemas	58
3.4.6.2.	Hipótese dos Blocos Construtivos	61
3.4.6.3.	A performance de AGCR	62
3.5.	Paralelização de algoritmos e GPGPU	62
4.	Materiais	71
5.	Metodologia.....	78
5.1.	Interface gráfica	79
5.2.	Pré-processamento dos dados	81
5.3.	Implementação de DRLSE em CUDA	83
5.4.	Comparações entre implementações de DRLSE	87
5.5.	Otimização dos parâmetros do algoritmo DRLSE	87
5.6.	Implementação dos algoritmos genéticos.....	89
5.7.	Avaliação de segmentação.....	90
5.7.1.	Implementação do coeficiente de Dice	91
5.7.2.	Implementação da Distância de Hausdorff	92
6.	Resultados e discussões	97
6.1.	Validação da métrica combinada.....	97
6.2.	Validação da implementação de DRLSE em CUDA e C++	99
6.3.	Otimização da implementação de DRLSE em CUDA	101
6.4.	Escolha de metaparâmetros para o algoritmo genético	105
6.5.	Varredura do espaço de parâmetros	119
6.6.	Exemplos de segmentações usando conjuntos de parâmetros gerados.	123
6.7.	Análise do gold-standard	127

6.8.	Análise de consistência	129
6.9.	Análise de sensibilidade	130
6.10.	Dependência no contexto de segmentação	134
6.11.	Uso total de memória em modos de treinamento e aplicação.....	138
6.12.	Análise em performance e complexidade computacional	139
6.12.1.	Performance de uma segmentação individual	139
6.12.2.	Performance do processo de treinamento	141
7.	Conclusões e extensões	143
7.1.	Publicações	144
7.2.	Trabalhos futuros.....	144
	Referências.....	146

1. INTRODUÇÃO

A segmentação de imagens é o processo de identificar e delinear objetos contidos nestas. É uma operação crucial em diversos ramos de computação baseada em imagens, como visualização 3D, filtragem, interpolação e registro, e engloba um enorme número de algoritmos e processos diferentes. A alta variabilidade nas características das imagens utilizadas, devido a diferentes metodologias de captura, processamento e exibição destas, fazem com que a escolha do correto algoritmo de segmentação e seus parâmetros a serem utilizados se torne uma tarefa crítica ao sucesso do procedimento (UDUPA, 2006).

No campo de Imagens Médicas, por meio de técnicas como raios-X, tomografia computadorizada (CT) e ressonância magnética (MR), torna-se possível a visualização da anatomia interna de pacientes. A segmentação das imagens obtidas é um estágio importante do processamento, que permite a delineação de órgãos e estruturas internas. A qualidade do algoritmo de segmentação usado melhora a qualidade da análise, do planejamento, do diagnóstico e permite o monitoramento eficiente do progresso de condições médicas (NOROUZI, 2014; TAHA, 2015).

Algoritmos de segmentação de imagens geralmente focam em uma de duas propriedades básicas de imagens: descontinuidade ou similaridade. Na primeira categoria, a abordagem visa particionar a imagem baseando-se em mudanças abruptas de intensidade, como bordas na imagem. Tais algoritmos também são classificados como “baseados em contornos”. Na segunda categoria, a abordagem principal é de particionar a imagem em regiões que são similares de acordo com algum critério. Estes algoritmos são frequentemente identificados como “baseados em regiões” (GONZALEZ e WOODS, 2002).

Técnicas de segmentação semiautomáticas, tanto baseadas em contorno como em regiões, permitem identificar e delinear algum objeto em uma imagem ou volume programaticamente a partir de uma entrada inicial providenciada pelo usuário, como um valor de brilho (ou cor) específico, um ou mais pontos iniciais pertencentes ao objeto (pontos sementes, ou *seeds*), ou um contorno inicial desenhado manualmente. Tais

técnicas são frequentemente aplicadas ao campo de Imagens Médicas devido à potencial capacidade de isolar tecidos e estruturas anatômicas, facilitando o diagnóstico e análise das imagens.

O algoritmo *Distance-Regularized Level-Set Evolution*, ou DRLSE (LI, 2010a), é uma técnica de segmentação baseada em contorno e, no contexto deste trabalho, usado como uma técnica semiautomática: o algoritmo evolui iterativamente um contorno inicial, delimitado manualmente, com base em equações diferenciais. Tal algoritmo é altamente paralelizável (fato cuja importância será discutida em sequência), apresenta boa robustez e é adaptável a diversos contextos. Essas características são acompanhadas, porém, pela necessidade da configuração manual de diversos parâmetros altamente sensíveis, junto com elevado custo computacional.

A determinação dos valores de parâmetros de algoritmos como este não é trivial, sendo necessário que o usuário possua conhecimento profundo da implementação e teoria por trás do algoritmo. Além disso, tal configuração normalmente consome tempo, envolve tentativa e erro, é feita de modo empírico ou heurístico, não garante nenhuma forma de otimalidade e pode levar ao comprometimento da acurácia e eficiência do algoritmo de segmentação (ABDUL-KARIM, 2005; HUTTER, 2007).

Nesse contexto foram desenvolvidos sistemas que automatizam a tarefa de escolha de parâmetros. Tais sistemas, frequentemente designados na literatura como Configuradores de Algoritmos, não apenas facilitam e agilizam a escolha de parâmetros, mas podem levar a melhores resultados (HUTTER, 2009; HUTTER, 2011) e proporcionar metodologias mais objetivas para comparação entre algoritmos. Inclusive, a presença de tais sistemas pode até vir a direcionar o próprio desenvolvimento destes algoritmos, de modo que se tornem mais parametrizados e adaptáveis a um maior número de situações, já que o fardo da escolha de parâmetros é gerenciado automaticamente (ANSÓTEGUI, 2009). Adicionalmente, a determinação automática dos parâmetros do algoritmo, especialmente no contexto de algoritmos de segmentação de imagens, faz com que seja possível, pelo menos em parte, aplicar o algoritmo sem conhecimento específico sobre a implementação ou teoria por trás deste, permitindo seu uso por profissionais de outras áreas, como por exemplo no contexto de diagnóstico médico.

O problema de configuração de algoritmos frequentemente é abordado como um problema de otimização, no qual uma função objetivo captura a performance do algoritmo alvo dependendo do grupo de parâmetros utilizado, e é minimizada ou maximizada (HUTTER, 2007).

Uma grande variedade de metodologias para configuradores de algoritmos pode ser encontrada na literatura. Em linhas gerais, tais metodologias podem ser classificadas em três diferentes categorias: Baseadas em gradientes, enumerativas e aleatórias (GOLDBERG, 1989). Adicionalmente, são documentados inúmeros outros algoritmos baseados em combinações de elementos destas metodologias.

Metodologias baseadas em gradientes utilizam as derivadas da função objetivo com respeito aos parâmetros do algoritmo alvo para tentar guiar a escolha dos melhores valores para tais parâmetros. Dificilmente tal procedimento é feito de modo analítico (ou indireto), já que o cálculo de tais derivadas depende da formulação de cada problema específico, sendo frequentemente difícil ou impossível (GOLDBERG, 1989). Metodologias diretas de exploração dos gradientes como *hill climbing* (GRATCH, 1992) ou *beam search* (MINTON, 1993) fazem a busca ao variar um ou mais parâmetros do algoritmo alvo e calculando os pontos correspondentes da função objetivo. Tais pontos são usados para aproximar seu gradiente, guiando a escolha de parâmetros às direções de gradiente crescente, “subindo a encosta” e eventualmente convergindo a algum tipo de máximo ou mínimo. Infelizmente, métodos puramente baseados em gradientes frequentemente ficam presos em máximos e mínimos locais ou regiões de baixa curvatura da função objetivo. Além disso, mesmo metodologias diretas não são capazes de lidar com funções objetivo altamente descontínuas ou irregulares (GOLDBERG, 1989).

Métodos enumerativos, também conhecidos como “busca exaustiva”, envolvem a exploração de todas as combinações possíveis de parâmetros para o algoritmo alvo. Tais abordagens são altamente ineficientes para algoritmos alvo com um grande número de parâmetros ou com elevado tempo computacional, como o caso da segmentação de imagens (GOLDBERG, 1989).

Métodos puramente aleatórios, por sua vez, apresentam eficiência excepcionalmente baixa, especialmente para otimização de algoritmos alvo com muitos parâmetros (GOLDBERG, 1989). Diversos algoritmos, porém, combinam elementos aleatórios com outras estratégias de busca. Alguns exemplos são algoritmos baseados em corridas (BIRATTARI, 2002; BIRATTARI, 2004; BALAPRAKASH, 2007), *simulated annealing* (KIRKPATRICK, 1983), ParamILS (HUTTER, 2009) e algoritmos genéticos (HOLLAND, 1975).

Os algoritmos genéticos apresentam alguns benefícios interessantes como configuradores de algoritmos alvo, quando comparados às demais abordagens levantadas (GOLDBERG, 1989):

- Funcionam com base em codificações dos parâmetros, de modo que são independentes dos domínios de tais parâmetros ou relações entre estes;
- Fazem a busca com uma população de amostras, impedindo que fiquem presos em mínimos ou máximos locais ou outros pontos da função objetivo;
- Avaliam performance puramente com base na função objetivo, não necessitando de informações adicionais como gradientes ou derivadas;
- Usam transições probabilísticas ao amostrar a função objetivo, de modo que apesar de não serem métodos puramente aleatórios, usam processos estocásticos para guiar a busca.

Tais características combinadas contribuem para a reconhecida robustez dos algoritmos genéticos (GOLDBERG, 1989). Tal robustez faz com que algoritmos genéticos sejam bons candidatos para a otimização de um algoritmo como DRLSE – os parâmetros do algoritmo DRLSE são altamente sensíveis, e a topologia do espaço de parâmetros é diferente para cada imagem e segmentação.

Além disso, algoritmos genéticos também são capazes de produzir bons resultados com menos avaliações do algoritmo alvo do que métodos de otimização baseados em busca exaustiva. Isso é importante para a otimização do algoritmo DRLSE, que apresenta alto custo computacional. Apesar deste benefício, o custo computacional total ainda é elevado (POSPICHAL, 2010).

Esse problema duplo, em que é necessário minimizar o custo computacional do algoritmo alvo simultaneamente a implementação de um configurador de parâmetros automático, pode ser abordado com técnicas para paralelização de algoritmos. Tais técnicas são especialmente relevantes no contexto deste trabalho, já que tanto algoritmos genéticos quanto o algoritmo de segmentação DRLSE são inerentemente paralelizáveis, significando que parte significativa do processamento destes pode ser realizada de modo independente e simultâneo.

Placas de vídeo, ou GPUs (*Graphics Processing Units*) são hardware inicialmente projetados para aceleração e paralelização de processos de renderização 3D. Por meio de plataformas relativamente recentes como Nvidia CUDA e OpenCL, tornou-se possível o uso destas para a aceleração de algoritmos para qualquer propósito (GPGPU), frequentemente alcançando ganhos em performance de milhares de vezes (POSPICHAL, 2010; DEBATTISTI, 2009). A plataforma CUDA pode, por exemplo, ser utilizada para a paralelização do algoritmo de segmentação de imagens DRLSE.

2. OBJETIVOS

Esse trabalho visa implementar um sistema para configuração automática dos parâmetros do algoritmo de segmentação de imagens *Distance-Regularized Level-Set Evolution* (DRLSE) (LI, 2010a) por meio do uso de algoritmos genéticos.

Graças à excelente capacidade de paralelização do algoritmo de segmentação DRLSE, esse trabalho tem como objetivo utilizar a plataforma CUDA para sua aceleração em GPU, potencialmente trazendo as etapas de configuração e segmentação ao regime interativo.

Mais especificamente, a partir de uma segmentação manual ou semiautomática *gold-standard* de um objeto em uma imagem bidimensional, o sistema configurará os parâmetros para que a segmentação proporcionada por DRLSE mais se aproxime de tal contorno. O processo de configuração objetivará maximizar métricas escolhidas para avaliação da segmentação gerada por DRLSE.

Após a configuração, os parâmetros determinados poderão ser usados para rapidamente segmentar imagens similares, por meio da evolução iterativa de um contorno aproximado delimitado manualmente. Os parâmetros obtidos também podem ser usados para segmentação automática de volumes, no qual um contorno convergido em apenas uma imagem pode ser usado como contorno inicial na segmentação das imagens vizinhas, progredindo em sequência às demais imagens da série.

3. FUNDAMENTAÇÃO TEÓRICA

Ao longo desta dissertação será utilizado o termo *spel* (do inglês, abreviação de *spatial element*) para se referir a elementos espaciais de imagens. Estes correspondem a pixels no caso de imagens bidimensionais e voxels no caso de imagens tridimensionais. As coordenadas de um *spel* c em tais imagens são descritas por $\vec{r}(c)$, sendo $\vec{r}(c) = (x, y)$ no caso de uma imagem bidimensional e $\vec{r}(c) = (x, y, z)$ no caso de uma imagem tridimensional. O valor descrito por cada *spel* será representado por $v(c)$, sendo c é um *spel* qualquer.

3.1. Imagens Médicas

Imagens médicas são aquelas que permitem a visualização das estruturas anatômicas e funcionais do paciente, possibilitando análise, diagnóstico e acompanhamento de modo não invasivo. Diversas técnicas foram desenvolvidas, cada uma com seus benefícios e desvantagens. Algumas destas modalidades, relevantes a esta dissertação, são discutidas abaixo.

Microscopia é a técnica do uso de lentes óticas para ampliação de objetos na escala de micrômetros (como células), permitindo a geração de imagens e visualização destes. Adicionalmente, contrastes podem ser utilizados: tratam-se de substâncias que naturalmente são concentradas em diferentes estruturas, e ao possuir cores marcantes ou fluorescência, destacam estruturas celulares contra o fundo, facilitando a visualização (MAIER, 2018).

Imageamento por raios-x é uma técnica utilizada há mais de um século. Esta envolve o uso de um emissor de raios-x de um lado do paciente, e um detector do lado oposto. Durante o exame, o emissor gera um pulso de curta duração, produzindo uma grande quantidade de raios-x. Uma grande fração destes interage com o paciente e uma parte o atravessa, atingindo o detector, onde uma imagem radiográfica é formada. Os tecidos internos do paciente absorvem a radiação em intensidades diferentes, fazendo com que

a distribuição homogênea de raios x que incide sobre o paciente se torne heterogênea ao deixá-lo em direção ao detector (BUSHBERG, 2012). Esta é uma técnica de imageamento baseada em projeções, de modo que cada ponto na imagem radiográfica no detector é uma somatória da informação ao longo de uma linha ao longo do paciente (BUSHBERG, 2012). Mesmo em doses baixas, raios-x são radiações ionizantes, que podem acarretar em danos ou destruição de células. Existe sempre a possibilidade de que tais danos possam acarretar em alterações genéticas e/ou incidência de câncer (SUETENS, 2009).

A tomografia computadorizada (CT) também envolve o uso de imageamento por raios-x, mas envolve a emissão e detecção de raios-x ao longo de múltiplos ângulos de incidência, frequentemente feito com a rotação do emissor e/ou detector ao redor do corpo do paciente. A informação obtida é combinada por meio de algoritmos computacionais, originando uma imagem precisa da anatomia interna tridimensional, sem sobreposição de tecidos (BUSHBERG, 2012). O processo é repetido conforme o detector se movimenta axialmente pela estrutura a analisar, gerando diversas seções transversais, ou “fatias”, ao longo do processo. Equipamentos modernos de CT são capazes de capturar centenas de imagens em apenas alguns segundos. Além disso, o uso de contrastes permite a análise funcional de diversos órgãos. Tais fatos, junto ao grande poder diagnóstico das imagens anatômicas geradas, fizeram com que CT substituísse a radiografia básica por raios-x em um grande número de aplicações (BUSHBERG, 2012). Apesar disso, ao usar uma grande quantidade de raios-x, alguns equipamentos de CT podem proporcionar uma dose de radiação de 10 a 100 vezes maior do que técnicas de imageamento por raios-x simples, de modo que a técnica deve ser utilizada com cautela (SUETENS, 2009).

A técnica de imageamento por ressonância magnética (MR) utiliza campos magnéticos e propriedades de ressonância magnética nuclear do próton (núcleo do átomo de hidrogênio, abundante em tecidos biológicos). O paciente é sujeito a fortes campos magnéticos e a diversos pulsos específicos de ondas de rádio. Os tecidos absorvem tais ondas e subsequentemente as reemitem, após um período de tempo que depende das propriedades magnéticas do tecido. Tais ondas reemitidas são captadas por antenas no detector, e a frequência e fase dos sinais obtidos permitem a

caracterização do tecido observado. (BUSHBERG, 2012). O resultado final é uma série de imagens de seção transversal, como CT. Além disso, scanners de MR permitem a realização de uma análise espectroscópica, possibilitando a visualização da composição bioquímica de tecidos em um dado volume. Tal fato, junto com a opção do uso de contraste, faz com que a técnica de MR também possa ser utilizada para análises funcionais (BUSHBERG, 2012; SUETENS, 2009).

A técnica de MR é de modo geral inofensiva, não usando radiação ionizante, e apresenta alta sensibilidade a variações anatômicas, especialmente entre alguns diferentes tipos de tecidos moles, de difícil visualização em exames de CT (SUETENS, 2009). Os exames de MR podem demorar dezenas de minutos, porém, e os fortes campos magnéticos usados fazem com que, caso seja necessário o uso de equipamento adicional (como marcapassos, grampos ou implantes), estes sejam especializados para operação em MR e usem materiais específicos. Tais desvantagens fazem com que, na maioria dos casos de trauma, exames de CT sejam preferidos (BUSHBERG, 2012).

Uma imagem médica bidimensional originada de uma modalidade computadorizada, como CT ou MR, consiste em um *grid* retangular de *spels*. Na maioria das modalidades, cada *spel* contém um único valor, que descreve algum tipo de propriedade de uma unidade de volume correspondente do paciente. Durante a visualização em um monitor, tais valores são convertidos à intensidade de luz, e interpretados por um profissional da área médica (BUSHBERG, 2012).

Cada imagem bidimensional pode ser visualizada individualmente, frequentemente em modo “pilha” (que se refere à ideia de empilhar as imagens da série umas sobre as outras), no qual a imagem visualizada pode ser rapidamente trocada pela imagem seguinte ou anterior pelo usuário, sendo possível acompanhar as estruturas exibidas ao longo das diversas imagens da série (BUSHBERG, 2012). Modernas técnicas de renderização volumétrica também permitem a exibição da informação de múltiplas imagens simultaneamente em modo 3D, seja por meio da reconstrução de superfícies ou de abordagens envolvendo projeções. Com isso torna-se possível a inspeção direta da cena em três dimensões, facilitando o planejamento e a extração de informação quantitativa das imagens, como dimensões de uma lesão (SUETENS, 2009).

A maioria da informação extraída de imagens médicas por profissionais de saúde tem caráter qualitativo e subjetivo, dependendo do profissional, das técnicas de renderização, brilho, contraste, o hardware usado para visualização e até das condições do ambiente. Com o intuito de se padronizar tal interpretação e uso de imagens médicas computadorizadas, e possibilitando o seu uso de modo objetivo em diagnóstico e terapêutica, o padrão *Digital Imaging and Communications in Medicine* (DICOM) foi criado. Originado de um comitê formado pelo *American College of Radiology* (ACR) e o *National Electrical Manufacturers Association* (NEMA), a primeira versão do padrão DICOM foi emitida em 1985. Este estabelece formatos de armazenamento e normas para padronização da transmissão e exibição de imagens médicas (MUSTRA, 2008; NEMA; BUSHBERG, 2012). Todas as imagens utilizadas nesta dissertação conformam-se ao padrão DICOM e, ao longo desta dissertação, o termo “imagem DICOM” é usado para se referir a uma imagem armazenada e exibida de acordo com as normas estabelecidas pelo padrão DICOM.

3.2. Segmentação de imagens

Um algoritmo de segmentação produz, a partir de uma imagem de entrada $I = \{c_1, c_2, \dots, c_n\}$, onde c_k indica um *spel*, uma partição $S_k = \{S_k^1, S_k^2, \dots, S_k^m\}$ em m diferentes subconjuntos de *spels* S_k^i , sendo $i \in [1, m]$. Ao longo desta dissertação serão consideradas apenas segmentações em que apenas dois subconjuntos são gerados em cada partição, S_k^1 e S_k^2 . Estas são também denominadas segmentações binárias.

É possível visualizar uma segmentação binária S_k com uma imagem binária B_{S_k} . Esta possui as mesmas dimensões de I , e cada um de seus *spels* $b_j \in B_{S_k}$ corresponde à um *spel* $c_j \in I$, sendo $j \in [1, n]$. Caso $c_j \in S_k^1$, temos que $v(b_j) = 1$, e caso $c_j \in S_k^2$, temos que $v(b_j) = 0$. Um exemplo de uma segmentação binária pode ser visualizado na Fig. 1.

3.2.1. Level Sets

Contornos ativos (também conhecidos como *snakes* ou *deformable models*), foram primeiro descritos por Kass et al, 1988, como curvas definidas no domínio de uma imagem que podem se mover devido a influência de forças internas (proveniente da curva em si) e externas (provenientes da influência da imagem), eventualmente convergindo em um alvo, como as bordas de um objeto. A ideia principal era de formalizar o problema como um problema de minimização de energia, de modo que a influência das forças guiasse a curva a um mínimo (ZHANG, 2013; WANG, 2014).

Figura 1 – Esquerda: Imagem 73, série 701-sT1W3DTFEAX PRE-30908, estudo 04-23-2000-MR BRAIN WOW CONTRAST-18837, paciente C3L-00629. Fonte: CPTAC, 2018a. Originalmente 528x528 pixels. Modalidade MR. Direita: Mesma imagem com uma segmentação binária.

As primeiras abordagens na área de contornos ativos originaram algoritmos que podem ser classificados como “contornos ativos paramétricos”. Nestes, a curva era definida explicitamente como função de um ou mais parâmetros. Tais abordagens tinham

como principais desvantagens a incapacidade de lidar com alterações topológicas (combinação e separação da curva em segmentos distintos), limitações na captura de objetos não-convexos, a impossibilidade de definir a energia a minimizar independentemente do esquema de parametrização, e a existência de instabilidades numéricas originadas do processo de minimização de energia (ZHANG, 2013; WANG, 2014).

Paralelamente, o método de *Level Sets* foi introduzido por Osher e Sethian em 1987 (OSHER, 1988) em um diferente contexto. Tal método define um contorno como o conjunto nível zero (*zero level set*) de uma função de maior dimensionalidade que o domínio da imagem, a função de *Level Sets* (LSF) (ZHANG, 2013; LI, 2010a). Apenas com os trabalhos de Caselles et al em 1993, Chopp em 1993 e Ruoy e Tourin em 1992, estabelecendo a classe de algoritmos de contornos ativos geométricos (também denominados contornos ativos geodésicos, ou GAC), a ideia de *Level Sets* foi trazida ao contexto de segmentação por contornos ativos (TAN, 2017; SHEN, 2002; SURI, 2002; XAVIER, 2005).

Contornos ativos geométricos visam então definir a curva em si de modo indireto: Para a segmentação de uma imagem n -dimensional, a função de *Level Sets* é definida como $\phi : \mathbb{R}^n \rightarrow \mathbb{R}$, e o conjunto nível zero de ϕ pode ser mais precisamente definido como o conjunto de *spels* c tal que $\phi(\vec{r}(c)) = 0$. Tal conjunto descreve uma curva no domínio da imagem, e a evolução de tal curva com o tempo é dada indiretamente pela evolução da função de *Level Sets*, de acordo com sua implementação. Note que é possível realizar uma segmentação com tal técnica devido ao fato de que o conjunto nível zero particiona a imagem em dois subconjuntos de *spels* que podemos denominar S_k^1 e S_k^2 , tal que um *spel* c pertence à S_k^1 se $\phi(\vec{r}(c)) \geq 0$, ou pertence à S_k^2 se $\phi(\vec{r}(c)) < 0$.

Devido a tal formulação indireta, contornos ativos geométricos também são reconhecidos por “contornos ativos implícitos” (ZHANG, 2013; LI, 2010a; WANG, 2014). Estes são livres de algumas das desvantagens características dos contornos ativos paramétricos, de modo que são capazes de lidar com alterações topológicas naturalmente, e podem ser calculados no *grid* cartesiano fixo da imagem, sem ser necessário parametrizar os pontos na curva (LI, 2010a; WANG, 2014).

Finalmente, é possível classificar algoritmos de contornos ativos geométricos em duas diferentes subclasses: Baseados em bordas (também conhecidos por baseados em equações diferenciais parciais), e baseados em regiões (também reconhecidos por baseados em modelos variacionais). GAC, e DRLSE (discutido em seguida), pertencem à primeira subclasse, enquanto a segunda subclasse contém algoritmos como um primeiro trabalho de Chan e Vese, 2001a (baseado em modelos de Mumford e Shah, 1989), e um segundo trabalho de Chan e Vese, 2001b, conhecido por *piecewise smoothing*. A principal discrepância entre as subclasses é o fato de que algoritmos da primeira não são capazes de segmentar o interior de objetos, sendo isso consequência da formulação de *Level Sets* usar apenas contornos fechados e ser influenciada apenas pelo gradiente local ao contorno. Algoritmos da segunda subclasse visam usar a informação estatística de todas as regiões da imagem (incluindo regiões internas) para impulsionar o processo de segmentação (ZHANG, 2013; APPIA, 2011). Apesar disso, é possível converter um algoritmo de uma subclasse em uma formulação da outra subclasse (ZHANG, 2013).

Apesar dos benefícios de contornos ativos geométricos em geral, o problema de instabilidade numérica ainda se mantinha presente. Geralmente, funções de *Level Set* são inicializadas com o formato de uma função distância, já que estas possuem diversas propriedades desejáveis ao processo de evolução (ZHANG, 2013; WANG, 2014). Ao longo de tal evolução da função de *Level Sets*, esta progressivamente desvia do formato inicial de função distância, originando irregularidades que podem levar a erros numéricos e na eventual destruição da estabilidade de todo o processo. Chopp et al foi o primeiro a propor o processo de “reinicialização” para remediar esse problema, no qual o processo de evolução da função de *Level Sets* é interrompido, esta é forçada ao formato de uma função distância, e o processo de evolução retomado (ZHANG, 2013; LI, 2010a; WANG, 2014).

Apesar do processo de reinicialização de fato remediar o problema, este apresenta diversas desvantagens e efeitos colaterais, como o potencial risco de impedir que novos contornos nível zero emergjam, e o risco de se erroneamente deslocar a localização da função de nível zero (ZHANG, 2013; LI, 2010a). Adicionalmente, o processo de reinicialização é um grave desacordo entre teoria e implementação: A evolução da curva

de *Level Sets* fundamentalmente não apresenta um componente de função distância, nem este é observado em soluções analíticas do processo, mas mesmo assim tal função é periodicamente e artificialmente forçada ao formato de uma função distância com a reinicialização. Finalmente, pode-se mencionar que não se sabe exatamente o momento apropriado de realizar tal reinicialização, que acaba sendo feito de modo *ad-hoc*. Tais fatos sugerem que a reinicialização deve ser usada o mínimo o possível (LI, 2010a).

Weber et al foi um dos primeiros a propor uma implementação de um modelo de contornos ativos geodésicos que dispensa a necessidade de um processo separado de reinicialização. Para isso, a função de *Level Sets* em cada passo de iteração é atualizada com um processo complexo envolvendo a solução de um problema de otimização, ao invés de um processo iterativo originado das equações de evolução da função de *Level Sets*, sendo novamente um desacordo entre teoria e implementação (LI, 2010a).

Um trabalho anterior de Li et al propôs um mecanismo para prevenir a reinicialização por meio de um termo na formulação variacional da função de *Level Sets* que penaliza desvios de um formato de função distância. Tal abordagem eliminou a necessidade de reinicialização, mas tinha o efeito colateral de prejudicar a acurácia numérica em algumas circunstâncias (LI, 2010a).

Li (2010) introduz uma nova formulação da técnica com termo de regularização, denominada *Distance-Regularized Level Set Evolution* (DRLSE). Esta formulação força a magnitude do gradiente da função de *Level Sets* a um de seus dois pontos mínimos ao longo do processo de evolução, mantendo o formato ideal de função distância especialmente nas proximidades do conjunto nível zero. O uso de tal termo de regularização completamente dispensa a necessidade de reinicialização, sem as desvantagens do trabalho anterior.

O funcional de energia minimizado no algoritmo DRLSE pode então ser descrito pela equação 1:

$$\mathcal{E}(\phi) = \mu \mathcal{R}_p(\phi) + \mathcal{E}_{ext}(\phi) \quad (1)$$

Na equação 1, $\mathcal{R}_p(\phi)$ representa o termo de regularização; $\mathcal{E}_{ext}(\phi)$ representa a energia relacionada às forças externas (ou seja, que dependem da imagem de entrada),

definida de tal maneira que é mínima quando o conjunto nível zero de ϕ se encontra na posição desejada; $\mu > 0$ é uma constante de peso; e o subscrito p se refere ao fato de que tal termo depende de uma energia potencial descrita por uma função $p: [0, \infty) \rightarrow \mathbb{R}$.

A equação 1 pode ser expandida na equação 2, que corresponde a descrição completa da funcional de energia minimizada pelo algoritmo no domínio da imagem Ω :

$$\mathcal{E}(\phi) = \mu \int_{\Omega} p(|\nabla \phi|) dx + \lambda \int_{\Omega} g \cdot \delta_{\varepsilon}(\phi) |\nabla \phi| dx + \alpha \int_{\Omega} g \cdot H_{\varepsilon}(-\phi) dx \quad (2)$$

Em 2, H_{ε} corresponde à uma aproximação da função de Heaviside. Os demais termos correspondem aos termos em 1, e serão descritos em seguida. Para detalhes da derivação do algoritmo de DRLSE, o leitor é direcionado ao trabalho original de Li et al, 2010. Para propósito desta dissertação, basta indicar que para o caso da segmentação de uma imagem bidimensional $I(x, y)$, o algoritmo DRLSE possui uma formulação iterativa completa descrita pelas equações de 3 a 7:

$$\phi_{x,y}^{k+1} = \phi_{x,y}^k + \tau \cdot \left[\begin{array}{c} \mu \cdot \text{div} \left(\frac{p'(|\nabla \phi_{x,y}^k|)}{|\nabla \phi_{x,y}^k|} \nabla \phi_{x,y}^k \right) + \\ \lambda \cdot \delta_{\varepsilon}(\phi_{x,y}^k) \cdot \text{div} \left(g(x, y) \cdot \frac{\nabla \phi_{x,y}^k}{|\nabla \phi_{x,y}^k|} \right) + \\ \alpha \cdot g(x, y) \cdot \delta_{\varepsilon}(\phi_{x,y}^k) \end{array} \right] \quad (3)$$

$$p(s) = \begin{cases} \frac{1}{(2\pi)^2} (1 - \cos(2\pi s)), & \text{se } s \leq 1 \\ \frac{1}{2} (s - 1)^2, & \text{se } s \geq 1 \end{cases} \quad (4)$$

$$p'(s) = \begin{cases} \frac{1}{2\pi} \sin(2\pi s), & \text{se } s \leq 1 \\ s - 1, & \text{se } s \geq 1 \end{cases} \quad (5)$$

$$g(x, y) = \frac{1}{1 + |\nabla G_{\sigma} * I(x, y)|^2} \quad (6)$$

$$\delta_{\varepsilon}(s) = \begin{cases} \frac{1}{2\varepsilon} \left[1 + \cos\left(\frac{\pi s}{\varepsilon}\right) \right], & |s| \leq \varepsilon \\ 0, & |s| > \varepsilon \end{cases} \quad (7)$$

Aqui, $\phi_{x,y}^k$ representa o valor da função de *Level Sets* para o *spel* c de coordenadas $\vec{r}(c) = (x, y)$ e instante k , e a variável τ indica o intervalo de tempo virtual entre iterações. Dentro dos colchetes, três termos são somados. Estes representam, respectivamente, o termo de regularização, o termo de comprimento e o termo de área, sendo $\mu > 0$, $\lambda > 0$

e α os respectivos pesos. Observa-se que o primeiro termo da soma se origina da derivação de $\mu\mathcal{R}_p(\phi)$ indicado na equação 1, enquanto que segundo e terceiro termos provêm da derivação de $\mathcal{E}_{ext}(\phi)$.

A função $p(s)$, sendo $p: [0, \infty) \rightarrow \mathbb{R}$ é a função escolhida para descrever energia potencial, e $p'(s)$ é sua derivada. $p(s)$ apresenta mínimos para $s = 0$ e $s = 1$, de modo que ao longo da iteração, ao minimizar $p(|\nabla\phi_{x,y}^k|)$, fazemos com que $|\nabla\phi_{x,y}^k|$ se aproxime do valor 1.0 próximo do conjunto nível zero de ϕ (característico de uma função distância), ou valor 0.0 (suavizando regiões distantes do conjunto nível zero de ϕ).

$g(x, y)$ é uma função indicadora de bordas, sendo projetada para que tenha valores mais baixos perto de bordas da imagem de entrada $I(x, y)$. Os baixos valores guiam a minimização da energia de ϕ , atraindo o conjunto nível zero. A convolução de $I(x, y)$ com um núcleo Gaussiano G com desvio padrão σ tem como efeito a suavização e remoção do ruído de $I(x, y)$. Em seguida obtém-se a magnitude do gradiente do resultado de tal convolução, produzindo o efeito total de detecção de bordas. Note que devido a propriedades da convolução, temos que $\nabla(G_\sigma * I(x, y)) = (\nabla G_\sigma) * (I(x, y))$, sendo também possível interpretar o denominador de $g(x, y)$ como a convolução com a derivada de um núcleo Gaussiano, sendo este um operador frequentemente usado para detecção de bordas em imagens (GONZALEZ e WOODS, 2002). Com a suavização, bordas abruptas da imagem são atenuadas, possibilitando a boa evolução de ϕ e atração do conjunto nível zero a partir de uma distância maior.

$\delta_\varepsilon(s)$ é a aproximação computável utilizada para a função delta de Dirac, sendo nesta dissertação o parâmetro ε , usado para ajuste do formato da aproximação, configurado como 1,5, como utilizado por padrão em LI, 2010a.

A convergência da formulação de Li *et al* é numericamente estável apenas caso seja satisfeita a condição de Courant-Friedrichs-Lewy, em que $\mu \cdot \tau < \frac{1}{4}$.

Finalmente, o efeito suavizador de $p(s)$ sobre $|\nabla\phi_{x,y}^k|$ em regiões distantes do conjunto nível zero faz com que $\phi_{x,y}^k$ permaneça relativamente constante ao longo das iterações. Tal fato proporciona a oportunidade de otimização de confinar os cálculos de

atualização de $\phi_{x,y}^k$ a uma região estreita ao redor do conjunto nível zero, considerando o restante como constante. Essa estratégia é denominada *Implementação em faixa estreita*, e o seu algoritmo pode ser observado em Li, 2006. Tal abordagem não é facilmente paralelizável, porém, e não será utilizada neste trabalho.

3.3. Avaliação de algoritmos e resultados de segmentação

A avaliação de algoritmos de segmentação e seus resultados é de grande importância na área médica, onde a eficácia do algoritmo pode acarretar em uma detecção mais precoce de alguma condição médica, ou em um monitoramento mais confiável. Além disso, como algoritmos de segmentação são frequentemente o primeiro passo da análise de informação de Imagens Médicas, é também essencial minimizar resultados errados e inapropriados (TAHA, 2015; FENSTER, 2005). Avaliar a confiabilidade e qualidade dos resultados produzidos por um algoritmo de segmentação é uma tarefa não trivial, porém, e inúmeros métodos foram desenvolvidos com tal objetivo.

Tais métodos podem ser primariamente classificados em dois grupos: O primeiro, definido como grupo dos métodos analíticos, envolve a avaliação teórica direta do algoritmo de segmentação em si, considerando seus princípios, suposições, propriedades, exigências, utilidades, complexidades, entre outros. Esse processo de avaliação independe da implementação, natureza ou objetivo da segmentação. O segundo grupo de métodos de avaliação envolve a análise empírica dos resultados obtidos, por meio do uso de imagens de teste ou medição objetiva da qualidade do resultado da segmentação, analisando assim o algoritmo de forma indireta (DLUGOSZ, 2009).

Métodos deste segundo grupo podem ser posteriormente classificados em outras duas classes: Os que usam da medição da qualidade e os que usam da medição da discrepância. Com a medição de qualidade, a segmentação é julgada sem conhecimento *a priori* de uma segmentação de referência, usando métricas como uniformidade dos segmentos ou diferença espectral entre os segmentos gerados (DLUGOSZ, 2009). Via

medição de discrepância, em contraste, o desempenho do algoritmo é avaliado ao comparar o resultado da segmentação com uma segmentação de referência. No caso de imagens reais, a segmentação de referência é obtida da mesma imagem de entrada. No caso de imagens sintéticas, a segmentação de referência pode ser obtida do seu processo de geração (DLUGOSZ, 2009).

Ao longo desta dissertação, duas metodologias serão usadas para avaliação dos resultados de segmentação. Ambas são metodologias empíricas, envolvendo análise da discrepância com relação a segmentações de referência. Tais metodologias são o coeficiente de Dice e a Distância de Hausdorff (HD). Tais métodos foram escolhidos em função de três critérios, dois deles função do trabalho realizado por Taha e Handbury, 2015:

- Ambos são facilmente paralelizáveis em GPGPU (ou seja, possuem parte significativa de sua computação passível de ser feita de modo independente e simultâneo);
- Possuem abordagens fundamentalmente diferentes, sendo o coeficiente de Dice baseado em sobreposição de áreas (classe 1) e HD baseado em distâncias espaciais (classe 6) (TAHA, 2015);
- Possuem correlação relativamente baixa (TAHA, 2015);

3.3.1. Coeficiente de Dice

O coeficiente de Dice, inicialmente desenvolvido para análise de associação ecológica entre diferentes espécies de organismos (DICE, 1945; TAHA, 2015), é a metodologia mais utilizada para validação de segmentações volumétricas (TAHA, 2015). Ao avaliar o coeficiente de Dice para uma segmentação S_a com respeito a uma segmentação de referência S_r , temos que o coeficiente pode ser formulado de acordo com a equação 8:

$$DICE = \frac{2|S_a^1 \cap S_r^1|}{|S_a^1| + |S_r^1|} \quad (8)$$

Onde $|S_a^1|$ indica a cardinalidade do primeiro subconjunto da segmentação S_a , ou seja, o número de *spe/s* contidos em S_a^1 . Temos então que $DICE \in [0, 1]$, apresentando valor mínimo (zero) quando S_a^1 e S_r^1 são disjuntos, e valor máximo (um) quando tais subconjuntos são equivalentes.

3.3.2. Distância de Hausdorff

Formalmente, a Distância de Hausdorff (HD) entre dois conjuntos de pontos A e B pode ser representada por:

$$HD(A, B) = \max(h(A, B), h(B, A)) \quad (9)$$

Sendo $h(A, B)$ denominada distância de Hausdorff direcionada, e é dada por:

$$h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\| \quad (10)$$

Onde $\|a - b\|$ representa algum tipo de norma, como a distância Euclidiana (TAHA, 2015; KRISHNAMURTHY, 2011).

A métrica de HD é, quando aplicada a nuvens de pontos, geralmente sensível a perturbações devido a valores anormais ou atípicos, de modo que são necessárias modificações ao algoritmo original para mitigar tal problema (TAHA, 2015). Ao longo desta dissertação, porém, HD será aplicado a segmentações originadas do algoritmo DRLSE, capaz de auto regularização e relativamente resistente à existência de valores atípicos, sendo possível a aplicação da versão original de HD. Adicionalmente o resultado de tais segmentações são contornos (conjunto nível zero), de modo que são naturalmente avaliados por uma métrica como HD.

3.4. Algoritmos genéticos

Algoritmos genéticos (AG) são métodos de busca ou otimização baseados nos mecanismos de seleção natural (GOLDBERG, 1989). Foram primeiramente postulados por John Holland, seus colegas e alunos na Universidade de Michigan em 1975 como

resultado de uma pesquisa que visava abstrair e explicar os processos adaptativos de sistemas naturais, bem como projetar sistemas artificiais que também possuísem tais características. Em seguida, pode-se citar a extensão feita por De Jong, 1975 para problemas de otimização, bem como a formulação matemática detalhada realizada por Goldberg, 1989. Atualmente, os AG pertencem ao grupo de Algoritmos Evolucionários, junto com os algoritmos de Estratégias Evolucionárias, Programação Evolucionária e Programação Genética (MOHANTA, 2012).

Diversos mecanismos utilizados em AGs são analogias a aspectos biológicos de populações sujeitas à seleção natural e evolução. Pontos no espaço de busca representam indivíduos, e uma população destes é controlada pelo algoritmo ao longo de diversas gerações. Indivíduos são removidos ou reproduzidos, originando descendentes, de acordo com a qualidade dos respectivos pontos no contexto da busca, avaliada conforme uma função de *fitness*.

Mais especificamente, o algoritmo codifica as coordenadas do ponto referente a um indivíduo em uma sequência de caracteres de tamanho e alfabeto finito. Boa parte dos autores se referem a tais sequências como cromossomos, mantendo a analogia biológica. Nesse caso, caracteres dos cromossomos correspondem a genes, e valores destes caracteres correspondem a alelos do gene (GOLDBERG, 1989).

Diversas metodologias de codificação podem ser utilizadas, sendo a codificação binária uma das alternativas mais canônicas. A Tab. 1 indica exemplos de codificações para pontos em um espaço de busca com três parâmetros, a_1 , a_2 e a_3 , sendo os três descritos por números inteiros de 0 a 3.

Apesar de cromossomos serem reinterpretadas como valores nos instantes de execução da função *fitness*, os AG manipulam os cromossomos com operações puramente textuais, apenas trocando e manipulando caracteres independentemente do que estes representem. Isso confere aos algoritmos genéticos significativa imparcialidade quanto ao domínio, topologia do espaço dos parâmetros ou suas respectivas derivadas (GOLDBERG, 1989).

Tabela 1 – Exemplos de codificação.

Parâmetros			Codificação para cromossomos	
a_1	a_2	a_3	Decimal	Binária
0	0	1	001	00 00 01
2	1	3	213	10 01 11
1	2	2	122	01 10 10

Alternativamente, ao invés de utilizar um alfabeto de números inteiros, algoritmos genéticos de codificação real (AGCR) possuem um número real em cada caractere, apresentando por exemplo um cromossomo como $s = (1.2, 0, -30.4)$. De modo geral, os operadores de seleção, recombinação, mutação e demais usados em AG comuns podem ser aplicados à AGCR com poucas alterações.

O uso de uma codificação real facilita a modelagem de problemas reais, nos quais parâmetros dificilmente tomam a forma de números inteiros. Em outras palavras, não é necessário mapear tais parâmetros reais à codificação em números inteiros, eliminando preocupações quanto à precisão, penhascos de Hamming ou adequação dos intervalos (ESHELMAN, 1993). Nessa dissertação serão usados exclusivamente AGCR.

Algoritmos genéticos podem ser decompostos em diversas fases, de acordo com a Fig. 2. As fases são descritas de modo mais detalhado nas seções 3.4.1 a 3.4.5.

3.4.1. Inicialização

A fase de inicialização de AGs envolve a geração de uma população aleatória de N indivíduos, correspondendo a escolha de pontos aleatórios no espaço de busca. Em seguida, alguma metodologia de codificação dos parâmetros de tais pontos é aplicada, originando uma população de cromossomos.

3.4.2. Avaliação de *fitness*

A função de *fitness* avalia o cromossomo respectivo de cada indivíduo, direcionando a seleção de um subconjunto destes, na etapa seguinte. A função escolhida é uma função objetivo, que descreve os alvos a serem atingidos pelo algoritmo genético. Na analogia com os processos naturais, *fitness* é a medida do quanto que um indivíduo é

adaptado ao seu ambiente, e indiretamente indica a quantidade de descendentes esperados de tal indivíduo – quanto maior o fitness, maior o número de descendentes (TOWNSEND, 2013).

Figura 2 – Fluxograma de um algoritmo genético. Fonte: Daniel Coelho

No contexto de otimização ou configuração de algoritmos de segmentação em imagens, pode-se configurar uma função de *fitness* que avalie o quão similar a segmentação obtida é de uma segmentação de referência *gold-standard*. A metodologia específica para tal avaliação depende da implementação, mas geralmente procura-se minimizar os erros mais relevantes para a aplicação. Se o contorno do segmento é o mais relevante, pode procurar minimizar a distância média quadrática entre este e um

contorno de referência, enquanto que se a região do segmento é mais relevante, pode procurar minimizar a diferença entre a intensidade média do objeto segmentado e a intensidade média de referência, ou a intersecção da área ou volume segmentado com a área ou volume de referência.

Em todas as etapas de avaliação de *fitness*, checa-se um critério de parada, interrompendo o AG quando o número de gerações ultrapassa um certo valor, todos os recursos computacionais foram consumidos ou um valor limiar de *fitness* foi alcançado. Caso contrário, o AG segue para a etapa de seleção.

3.4.3. Seleção

Em linhas gerais, a seleção envolve a determinação de um subconjunto de cromossomos com *fitness* satisfatória. Os cromossomos deste subconjunto, também denominado *mating pool* ou população intermediária, correspondem aos indivíduos da população que gerarão descendentes.

Diversas metodologias de seleção são documentadas na literatura. Algumas operam diretamente sobre os valores de *fitness*, enquanto que algumas funcionam com base em valores derivados dos valores de *fitness*. Canonicamente, usa-se o método da roleta ou seleção por *fitness* proporcional, no qual cada valor de *fitness* f_i , referente ao cromossomo i , é convertido em um valor de probabilidade de seleção de i , P_i , de acordo com a equação 11:

$$P_i = \frac{f_i}{\sum_i^N f_i} \quad (11)$$

Se a população intermediária deve ter M indivíduos, M seleções independentes são realizadas, sendo que em cada cromossomo i é selecionado com probabilidade P_i . Nota-se que um mesmo cromossomo i pode ser selecionado múltiplas vezes. Além disso, apesar de ser mais provável que um cromossomo com *fitness* maior seja selecionado, ainda é possível, porém improvável, que cromossomos com *fitness* baixo sejam selecionados (GOLDBERG, 1989).

Pressão seletiva é definida como a intensidade com que o processo de seleção favorece indivíduos com maior *fitness* (MILLER, 1995). Para a seleção por *fitness* proporcional, por exemplo, conforme o AG progride e passa a convergir em uma solução global satisfatória, a variabilidade de valores de *fitness* se reduz, de modo que o processo de seleção passa a ser menos eficaz na seleção dos melhores indivíduos. Isso pode ser interpretado como uma redução da pressão seletiva, não sendo constante ao longo das gerações (WHITLEY, 1994).

Apesar de ser possível controlar a pressão seletiva para a seleção proporcional com alguns fatores de escala (WHITLEY, 1994; BÄCK, 1996), outras metodologias são aplicadas na prática, como a seleção por torneio ou seleção por ranking, que apresentam pressão seletiva constante e controlável. O controle da pressão seletiva é essencial, pois grandemente influencia a taxa de convergência do AG. Valores muito baixos podem implicar em tempo de convergência desnecessariamente longo, enquanto que valores muito altos podem implicar em convergência prematura em soluções não ótimas (MILLER, 1995).

Após a seleção, os indivíduos da população intermediária são pareados dois-a-dois aleatoriamente, e uma cópia de cada cromossomo do par é gerado, correspondendo aos descendentes do mesmo. Estes cromossomos filhos são sujeitos a diversos operadores evolucionários. Os dois mais frequentemente utilizados são recombinação e mutação (GOLDBERG, 1989).

Uma exceção a tal regra é a técnica de seleção elitista. Aplicada paralelamente à uma outra técnica de seleção, consiste em transmitir um ou mais dos indivíduos com maiores *fitness* de uma geração diretamente à geração seguinte (BALUJA, 1995).

3.4.3.1. Seleção por torneios

A técnica de seleção por torneios envolve selecionar T_o indivíduos aleatórios da população e, dentre eles, escolher o indivíduo de maior *fitness*. O processo (um torneio) é repetido quantas vezes o necessário, até preencher a população intermediária (GOLDBERG, 1991a).

3.4.3.2. Seleção por ranking linear

A técnica de seleção por ranking linear foi inicialmente sugerida por Baker, 1985 para eliminar as desvantagens da técnica básica de seleção proporcional, na qual os indivíduos são atribuídos probabilidade de seleção proporcional aos seus valores de *fitness* (BLICKLE, 1995; WHITLEY, 1989).

Na seleção por ranking linear, os N indivíduos são ordenados de acordo com os valores de *fitness*, e o ranking 1 é atribuído ao melhor indivíduo, enquanto que N atribuído ao pior indivíduo. A probabilidade de seleção de cada indivíduo i , P_i em função do ranking r_i é então calculada de acordo a equação 12, sendo $\frac{\eta^-}{N}$ a probabilidade do pior indivíduo ser selecionado, e $\frac{\eta^+}{N}$ a probabilidade do melhor indivíduo ser selecionado. Note que as condições $1 \leq \eta^+ \leq 2$ e $\eta^- = 2 - \eta^+$ devem ser obedecidas.

$$P_i = \frac{1}{N} \left(\eta^+ - (\eta^+ - \eta^-) \cdot \frac{r_i - 1}{N - 1} \right); \quad r_i \in \{1, \dots, N\} \quad (12)$$

Para melhor pressão seletiva, é recomendado usar $\eta^+ = 1,1$, e então temos que $\eta^- = 0,9$. (BÄCK, 1996; BAKER, 1985).

A Tab. 2 indica um exemplo de seleção por ranking linear para uma população de 5 indivíduos. Note que a probabilidade de seleção varia linearmente entre os indivíduos, apesar dos valores de *fitness* variarem de modo não linear (ou não variarem).

Tabela 2 – Seleção por ranking linear. Fonte: Daniel Coelho

Indivíduo	<i>Fitness</i>	Ranking	Probabilidade de seleção (%)
A	12,30	1	22,00
B	12,10	2	21,00
C	3,00	3	20,00
D	3,00	4	19,00
E	0,00	5	18,00
Total:			100,00

As principais vantagens deste operador de seleção é que despreza a necessidade de escalonar os valores de *fitness* artificialmente, é resistente contra convergência

prematura e apresenta bom controle da pressão seletiva ao configurar o parâmetro η^+ (BÄCK, 1996; BAKER, 1985).

3.4.3.3. Seleção por ranking exponencial

Similar ao operador de seleção por ranking linear, os N indivíduos são ordenados de acordo com os valores de *fitness*, e o ranking $r_i = 1$ é atribuído ao melhor indivíduo i , enquanto que $r_i = N$ atribuído ao pior indivíduo. Na seleção por ranking exponencial, o valor de ranking é usado como expoente para uma base $b \in (0,1)$, de modo que a probabilidade de seleção de um indivíduo i , P_i , em função de seu ranking r_i é dada pela equação 13, modificada a partir da formulação de BLICKLE, 1995. Quanto menor b , mais exponencial se torna a distribuição (BLICKLE, 1995):

$$P_i = \frac{b - 1}{b^N - 1} b^{r_i - 1}; \quad r_i \in \{1, \dots, N\} \quad (13)$$

A Tab. 3 indica um exemplo de seleção por ranking exponencial.

Tabela 3 – Seleção por ranking exponencial. Fonte: Daniel Coelho

Indivíduo	<i>Fitness</i>	Ranking	P_i para $b = 0,5$ (%)	P_i para $b = 0,2$ (%)
A	12,30	1	51,61	80,03
B	12,10	2	25,81	16,01
C	3,00	3	12,90	3,20
D	3,00	4	6,45	0,64
E	0,00	5	3,23	0,13
Total:			100,00	100,00

3.4.3.4. Amostragem universal estocástica

Desenvolvido por Baker, 1987, o operador de seleção amostragem universal estocástica (*Stochastic Universal Sampling*, ou SUS) trata-se de um método de seleção eficiente, com zero viés de seleção e mínimo espalhamento.

Viés de seleção é definido como a diferença absoluta entre a teórica probabilidade de seleção de um indivíduo e a real taxa de amostragem observada. Baixo viés de seleção indica então alta acurácia do processo de seleção (BAKER, 1987).

Seja $e_i \in \mathbb{R}$ o número esperado de descendentes para um indivíduo i em uma dada geração, e $d_i \in \mathbb{N}$ o número de descendentes observado. O espalhamento é definido como o intervalo de possíveis valores para d_i , e indica a consistência do processo de seleção. O espalhamento mínimo é definido como o mínimo espalhamento possível que teoricamente permite zero viés de seleção, ou seja, é o espalhamento em que vale a equação 14 (BAKER, 1987). Na equação 14, $\lfloor \cdot \rfloor$ e $\lceil \cdot \rceil$ são os operadores piso e teto, respectivamente.

$$d_i \in [\lfloor e_i \rfloor, \lceil e_i \rceil] \quad (14)$$

Na Fig. 3, 6 indivíduos (A, B, C, D, E e F) são dispostos em um vetor, com tamanhos proporcionais aos seus valores de *fitness*. O valor total de *fitness* é então normalizado ao intervalo $[0,1]$. O método de seleção proporcional, por exemplo, envolve sortear um número aleatório $rand \in [0,0, 1,0]$ para cada seleção, e escolher o indivíduo no vetor que corresponde à localização de $rand$ (indicados por setas na Fig. 3. Apenas um valor de $rand$ é indicado para clareza).

Figura 3 – Visualização do processo de seleção proporcional. Fonte: Daniel Coelho

Em contraste, a amostragem universal estocástica pode ser visualizada com o esquema da Fig. 4. Para o processo, um único número aleatório $rand$ é sorteado, e os M indivíduos da população intermediária são obtidos ao se amostrar o vetor, a partir de $rand$, em intervalos uniformes espaçados em $1/M$.

Figura 4 – Visualização do processo de seleção por amostragem universal estocástica. Fonte: Daniel Coelho

Note que tanto para seleção proporcional quanto para amostragem universal estocástica, não é necessário ordenar os vetores antes da seleção, apesar de as Fig. 3 e 4 apresentarem situações em que isso foi feito (BAKER, 1987).

3.4.3.5. Seleção por truncamento

Na seleção por truncamento com um limítrofe T_r , apenas a fração $T_r \in [0, 1]$ dos melhores indivíduos pode ser selecionada, e todos são atribuídos a mesma probabilidade de seleção. Esse método foi introduzido por Mühlenbein e Schlierkamp-Voosen em 1993 (BLICKLE, 1995).

3.4.4. Recombinação

Recombinação ou *crossover*, também parte da analogia biológica, é um processo no qual ocorre uma troca de informação entre os cromossomos de um par, originando novos cromossomos distintos, combinações dos iniciais. Para cada pareamento de cromossomos na população intermediária, a recombinação é realizada com probabilidade $P_c \in [0, 1]$. Os métodos canônicos de recombinação são ponto único, dois pontos e uniforme (WHITLEY, 1994).

Recombinação de ponto único envolve a escolha aleatória de uma localização l nos cromossomos originais. Todos os caracteres após l são trocados entre os dois cromossomos. Recombinação de dois pontos é similar, mas duas localizações diferentes são selecionadas, e o segmento de caracteres entre eles é trocado entre os

cromossomos. Recombinação uniforme envolve criar novos cromossomos em que cada caractere é escolhido separadamente e aleatoriamente a partir dos dois cromossomos pais (WHITLEY, 1994). Os três modelos de recombinação canônicos são esquematizados na Fig. 5.

Cada uma das metodologias de recombinação supracitadas têm vantagens e desvantagens. Recombinação uniforme perturba mais os cromossomos, explorando mais intensamente o espaço de busca, enquanto que recombinação de um ou dois pontos causa menor perturbação nos cromossomos, explorando o espaço de busca de modo mais conservador, mas trocando caracteres com viés à localização destes nos cromossomos. Na recombinação de ponto único, por exemplo, é mais provável que um caractere mais à direita no cromossomo seja trocado por recombinação do que um caractere mais à esquerda (WHITLEY, 1994).

Figura 5 – Diferentes metodologias de recombinação. Os cromossomos originais (acima) são recombinaados, originando novos cromossomos (abaixo). Fonte: Daniel Coelho

Outras metodologias apresentam bons resultados com o uso de mais de dois cromossomos originais, ou usando a informação de um cromossomo para alteração da sequência ou valores da outra, sem a transmissão direta de informação (EIBEN, 1994; TING, 2005).

3.4.4.1. Recombinação aritmética

Recombinação aritmética é usada principalmente em AGCR, e é definida como a interpolação linear dos dois cromossomos pais, usando um único número aleatório $rand \in [0,1]$. Sejam $\vec{a} = (a_1, a_2, \dots, a_n)$ e $\vec{b} = (b_1, b_2, \dots, b_n)$ dois indivíduos de um AGCR com n parâmetros reais por indivíduo. O operador de recombinação aritmética gera dois descendentes $\vec{a}'$ e $\vec{b}'$, cujos parâmetros são dados por:

$$a'_i = rand \cdot a_i + (1 - rand) \cdot b_i \quad i \in \{1, \dots, n\} \quad (15)$$

$$b'_i = (rand - 1) \cdot a_i + rand \cdot b_i \quad i \in \{1, \dots, n\} \quad (16)$$

Esse operador explora pontos no espaço de parâmetros pertencentes à um segmento de reta conectando $\vec{a}$ e $\vec{b}$. Michalewicz usa o termo recombinação aritmética para tal operador (MICHALEWICZ, 1996), mas tal operador é comum, e recebe diferentes nomes dependendo do autor, como “recombinação intermediária” ou “recombinação linear” (MICHALEWICZ, 1994).

3.4.4.2. Recombinação BLX- α

Desenvolvido por Eschelman e Shaffer, 1993, trata-se de uma generalização da interpolação linear dos parâmetros dos dois indivíduos pais. Difere da recombinação aritmética em dois aspectos: Ao usar um diferente número aleatório para cada parâmetro; e ao permitir que os valores gerados para cada gene descendente se encontrem fora do intervalo dos valores originais de cada gene.

Para dois indivíduos a e b de um AGCR, a Fig. 6 indica em amarelo, para dois parâmetros, a região do espaço de parâmetros que em BLX- α é capaz de gerar descendentes. As equações de 17 a 19 descrevem a formulação do BLX- α , sendo os indivíduos a', b' são gerados a partir do cruzamento de a e b .

$$\Delta_i = |a_i - b_i| \quad i \in \{1, \dots, n\} \quad (17)$$

$$a'_i = rand([\min(a_i, b_i) - \alpha_{BLX}\Delta_i, \max(a_i, b_i) + \alpha_{BLX}\Delta_i]) \quad i \in \{1, \dots, n\} \quad (18)$$

$$b'_i = rand([\min(a_i, b_i) - \alpha_{BLX}\Delta_i, \max(a_i, b_i) + \alpha_{BLX}\Delta_i]) \quad i \in \{1, \dots, n\} \quad (19)$$

Figura 6 – Espaço de parâmetros varrido pelo operador de recombinação BLX- α aplicado à dois indivíduos a e b . Fonte: Daniel Coelho

Eschelman e Shaffer recomendam o uso de $\alpha_{BLX} = 0,5$, obtendo uma situação similar à Fig. 6. Nesta configuração, a região amarela confere a tal operador a capacidade de explorar o espaço de parâmetros de um modo que outros operadores de recombinação não são capazes, sendo de certo modo uma combinação entre recombinação e mutação. Além disso, o intervalo dos possíveis valores para um parâmetro depende da distância entre os valores do parâmetro nos indivíduos pais. Tal comportamento confere ao BLX- α uma característica de focalizar a busca ao longo do progresso do AGCR (ESHELMAN, 1993).

3.4.4.3. Recombinação binária simulada

O operador de recombinação binária simulada (SBX), (DEB, 1995) busca emular em AGCR algumas das propriedades que a recombinação de um ponto apresenta quando aplicada à AG de codificações binárias, sendo a principal que os valores dos genes descendentes apresentam a mesma distância ao valor médio dos genes dos pais.

Adicionalmente, possui outras propriedades interessantes, como o fato de que valores próximos aos pais são preferidos monotonicamente (DEB, 2007).

Sejam $\vec{a} = (a_1, a_2, \dots, a_n)$ e $\vec{b} = (b_1, b_2, \dots, b_n)$ dois indivíduos de um AGCR com n parâmetros reais por indivíduo, e $\vec{a}' = (a'_1, a'_2, \dots, a'_n)$ e $\vec{b}' = (b'_1, b'_2, \dots, b'_n)$ os indivíduos da população intermediária gerados por SBX. Temos então que SBX garante a seguinte propriedade:

$$\frac{a'_i + b'_i}{2} = \frac{a_i + b_i}{2}$$

Seja $rand \in [0,1]$ com distribuição uniforme. Temos então que o operador SBX pode ser descrito pela seguinte formulação:

$$\beta_i = \begin{cases} (2 \cdot rand)^{\frac{1}{\eta_{SBX}+1}} & \text{se } rand \leq \frac{1}{2} \\ \left(\frac{1}{2(1-rand)}\right)^{\frac{1}{\eta_{SBX}+1}} & \text{se } rand > \frac{1}{2} \end{cases}$$

$$a'_i = \frac{1}{2}[(1 + \beta_i)a_i + (1 - \beta_i)b_i]$$

$$b'_i = \frac{1}{2}[(1 - \beta_i)a_i + (1 + \beta_i)b_i]$$

Na equação 20, β_i corresponde ao fator de espalhamento, descrevendo a razão do espalhamento dos genes dos indivíduos descendentes em relação ao espalhamento dos genes dos indivíduos pais, ou seja:

$$\beta_i = \left| \frac{a'_i - b'_i}{a_i - b_i} \right| \quad (20)$$

O parâmetro η_{SBX} , denominado “índice de distribuição” descreve o formato do fator de espalhamento β_i , de modo que valores altos tendem a gerar descendentes próximos dos pais, e valores baixos permitem a geração de descendentes distantes dos pais. O valor ótimo para a constante η_{SBX} depende do problema a otimizar, mas valores comuns

são 0, 2 ou 5. Especialmente $\eta_{SBX} = 5$ parece mostrar bom balanço entre exploração e exploração para diversos problemas (DEB, 1995; DEB, 2001; ORTIZ-BOYER, 2007).

3.4.5. Mutação

Mutação é o operador evolucionário aplicado em seguida. Análogo ao fenômeno genético de mesmo nome, envolve alterar de modo aleatório caracteres dos cromossomos descendentes com alguma probabilidade P_M , geralmente com valores abaixo de 1%. A principal função desse operador é a de garantir que nenhum valor de um caractere (um alelo) seja acidentalmente e permanentemente excluído da população (convergência prematura), sempre proporcionando uma chance que este retorne por meio de mutações (GOLDBERG, 1989; WHITLEY, 1994).

3.4.5.1. Mutação uniforme

Seja $\vec{a} = (a_1, a_2, \dots, a_n)$ um indivíduo com n parâmetros reais em um AGCR. O operador de mutação uniforme gera um número aleatório $rand \in [0, 1]$ para cada caractere a_i , $i \in \{1, \dots, n\}$. Se $rand \geq P_M$, a_i é substituído por um novo valor aleatório, a'_i , gerado com distribuição uniforme a partir do intervalo de valores possíveis para a_i (MICHALEWICZ, 1996).

3.4.5.2. Mutação gaussiana

O operador de mutação gaussiana é uma simples variação do operador de mutação uniforme. Para um indivíduo $\vec{a} = (a_1, a_2, \dots, a_n)$ de um AGCR, esse operador gera um número aleatório $rand \in [0, 1]$ para cada caractere a_i , $i \in \{1, \dots, n\}$. Se $rand \geq P_M$, a_i é substituído por um novo valor aleatório, a'_i . O valor atual de a_i é usado como centro, e junto com um desvio padrão σ_m , a equação 21 descreve a função de densidade de probabilidade gaussiana para potenciais novos valores de a'_i .

$$P_{a'_i} = \frac{1}{\sigma_m \sqrt{2\pi}} \cdot e^{-\frac{(a'_i - a_i)^2}{2\sigma_m^2}} \quad (21)$$

Algumas implementações também estabelecem um intervalo de valores aceitáveis, e grameiam o valor aleatório produzido à tal intervalo (HINTERDING, 1995; KOENIG, 2002, MATEO, 2012; BEYER, 2002).

3.4.6. Eficiência e eficácia de algoritmos genéticos

À primeira vista, é difícil visualizar como um processo essencialmente estocástico, como a evolução de um algoritmo genético, pode proporcionar uma varredura satisfatória do espaço de soluções. Tentativas de justificar matematicamente a eficiência e eficácia de algoritmos genéticos se baseiam em duas principais ideias: A teoria dos esquemas, e a hipótese dos blocos construtivos.

3.4.6.1. Teoria de Esquemas

Proposta por Holland em 1975, a teoria de esquemas é uma tentativa de descrever como que um algoritmo genético converge em soluções ao amostrar hiperplanos do espaço de parâmetros.

Um esquema H consiste em um subconjunto do conjunto de todos os cromossomos possíveis para a codificação e formulação do problema. Por exemplo, em um problema no qual cromossomos possuem alfabeto binário e 5 caracteres, um esquema $H1$ pode ser descrito pela expressão $H1 = (0, 1, *, *, *)$. Tal esquema descreve o subconjunto de cromossomos que apresentam 0 em sua primeira posição, 1 na segunda posição, e indica com o símbolo $*$ que o valor nas demais posições pode ser qualquer (BÄCK, 1996). Tal esquema descreve então o seguinte subconjunto de cromossomos:

$(0, 1, 0, 0, 0)$	$(0, 1, 0, 0, 1)$	$(0, 1, 0, 1, 0)$	$(0, 1, 0, 1, 1)$
$(0, 1, 1, 0, 0)$	$(0, 1, 1, 0, 1)$	$(0, 1, 1, 1, 0)$	$(0, 1, 1, 1, 1)$

Na teoria dos esquemas, cada esquema descreve um hiperplano no espaço de parâmetros. Essa representação pode ser visualizada facilmente para o caso tridimensional: Considere um espaço de parâmetros com três dimensões, codificado então por cromossomos binários de $n = 3$ caracteres. A disposição de tais cromossomos

neste espaço pode ser visualizada na Fig. 7. Note também o hiperplano preenchido em azul (no caso um simples plano bidimensional), descrito pelo esquema $H2 = (0,*,*)$.

Figura 7 – Espaço de parâmetros com cromossomos binários de 3 caracteres. Em azul, o hiperplano descrito pelo esquema $H2 = (0,*,*)$. Fonte:

Daniel Coelho

Cada esquema em si representa um hiperplano do espaço de parâmetros, e cada cromossomo pertence à $2^n - 1$ hiperplanos. Conforme o algoritmo genético progride, os cromossomos da população amostram os diversos hiperplanos do espaço de parâmetros de modo implícito e paralelo. Entende-se então que tais hiperplanos (ou esquemas) competem por representatividade na população. O efeito cumulativo de paralelamente avaliar tais competições implícitas entre os esquemas amostrados pelos cromossomos foi denominado por Holland como “paralelismo implícito”. Adicionalmente, Holland mostrou como uma população de N indivíduos amostra de modo útil no mínimo N^3 esquemas ao longo de sua evolução. O modo com que o paralelismo implícito afeta o progresso do algoritmo genético pode ser compreendido pela formulação precisa da teoria dos esquemas, discutida em seguida.

Seja a ordem $Ord(H)$ de um esquema dada pelo número de posições fixas. Temos então $Ord(H1) = 2$ e $Ord(H2) = 1$.

O comprimento definitório de um esquema H corresponde a distância entre o primeiro e último caracteres diferentes de *. Para o esquema $H3 = (*, 1, 0, *, *, 1, *, *)$ temos comprimento definitório $\Delta(H) = 6 - 2 = 4$. Tal métrica indica o número de diferentes posições possíveis para um ponto de recombinação que perturbe a parte significativa do esquema, ou seja, sugere o nível de compactação da informação contida no esquema (WHITLEY, 1994).

O *fitness* de um esquema H na geração t , dada por $f(H, t)$ pode ser definido como a média dos valores de *fitness* para todos os cromossomos S que pertencem ao esquema H , na geração t , de acordo com a equação 22:

$$f(H, t) = \sum_{S \in H} f(S) \quad (22)$$

Usando as grandezas descritas acima, pode-se modelar como o número de cromossomos pertencentes a um certo esquema H na geração t , descrito por $\xi(H, t)$ e denominado representatividade de H , varia conforme o algoritmo genético progride. Tal modelo é dado pela inequação 23, que descreve o crescimento reprodutivo do esquema H . Sua formulação precisa depende dos operadores usados, mas para um caso simples no qual são usados seleção proporcional, recombinação de 1 ponto e mutação pontual, temos:

$$\xi(H, t + 1) \geq \frac{\xi(H, t) \cdot f(H, t)}{\bar{f}(t)} \left[1 - P_c \frac{\Delta(H)}{n - 1} - P_M \cdot Ord(H) \right] \quad (23)$$

Na inequação 23, note que n descreve o tamanho dos cromossomos e $\bar{f}(t)$ descreve o *fitness* médio dentre os cromossomos de todos os esquemas na geração t . Além disso, $P_c \in [0, 1]$ descreve a probabilidade de recombinação e $P_M \in [0, 1]$ descreve a probabilidade de mutação.

Note como do lado direito da inequação 23, o termo fora dos colchetes descreve o efeito da seleção proporcional: A representatividade de H , $\xi(H, t)$, é ponderada pelo *fitness* de H quando comparado ao *fitness* médio da população na geração t . O termo dentro dos colchetes indica a probabilidade de sobrevivência de H , indicando o efeito da

recombinação ponderado pelo comprimento definitório $\Delta(H)$, e o efeito da mutação ponderado pela ordem $Ord(H)$.

A inequação 23 pode ser interpretada de acordo com a definição formal do Teorema dos Esquemas, que diz que esquemas H com $\Delta(H)$ curto, $Ord(H)$ baixo e $f(H, t)$ acima da média têm um aumento exponencial de representatividade em gerações consecutivas de um algoritmo genético. Temos então que conforme o algoritmo genético progride, diversos hiperplanos são amostrados paralela e implicitamente e, conforme estes competem por representatividade, o *fitness* da população como um todo tende a aumentar.

Apesar de uma boa ferramenta para visualização do mecanismo de amostragem de hiperplanos, a teoria dos esquemas como descrita nesta dissertação possui diversas limitações e críticas. Whitley, 1991, menciona o fato de apresentar apenas informação limitada sobre uma ou poucas gerações no futuro, além de indicar o trabalho de Grefenstette e Baker, 1989 que indica que o *fitness* de um hiperplano como um todo também só é relevante a poucas gerações após o início da evolução do algoritmo. Após tal período, o *fitness* de cada esquema ou hiperplano varia drasticamente devido ao viés de seleção. Altenberg, 1995 descreve como que a teoria dos esquemas não aborda o componente de busca de algoritmos genéticos, e não é capaz de distinguir entre problemas em que algoritmos genéticos são eficientes dos problemas em que não são.

Outros trabalhos mais recentes modificam a teoria dos esquemas na tentativa de responder e resolver tais críticas (WHITE, 2009; POLI, 2000; POLI, 2003). O campo da teoria dos esquemas é altamente estudado até hoje, e trabalhos mostrando novas formulações e aplicações são publicados anualmente.

3.4.6.2. Hipótese dos Blocos Construtivos

Consequência da teoria dos esquemas, a hipótese dos blocos construtivos diz que um algoritmo genético busca desempenho quase-ótimo por meio da justaposição de esquemas curtos, de baixa ordem e alto *fitness*, denominados blocos construtivos (GOLDBERG, 1989; WHITE, 2014).

3.4.6.3. A performance de AGCR

A teoria dos esquemas apresenta diversos argumentos a favor do uso de alfabetos de baixa cardinalidade, sugerindo que AGCR apresentariam baixa eficiência. Em contrapartida, diversos trabalhos desde 1957 até 1989 utilizavam alguma forma de AGCR com sucesso significativo, originando uma discrepância entre teoria e prática. Apenas em 1991, alguns trabalhos começaram a abordar tal discrepância e proporcionaram certa corroboração da eficiência observada empiricamente nos AGCR (GOLDBERG, 1991b).

De acordo com tais trabalhos, AGCR apresentam eficiência superior por diferentes motivos, como a maior variedade de operadores aplicáveis e a maior facilidade de incorporar informação específica ao problema (DAVIS, 1991). Goldberg, 1991b busca justificar a eficiência de AGCR ao sugerir que alfabetos virtuais de baixa cardinalidade, que facilitam a amostragem de hiperplanos, surgem do uso de alfabetos de alta cardinalidade. Além disso, Radcliffe, 1991 e Wright, 1991 apresentam outras justificativas e argumentos a favor de AGCR.

Apesar de tais trabalhos, a discrepância entre teoria e prática continua até hoje, também função das discussões sobre a validade da própria teoria dos esquemas. Por outro lado, existem também tentativas de adaptar a teoria dos esquemas de modo a explicar a eficiência de AGCR, como em Eschelman e Shaffer, 1993.

3.5. Paralelização de algoritmos e GPGPU

A maioria dos algoritmos são sequenciais, ou seja, indicam uma sequência de passos a serem executados, sendo cada passo traduzido em instruções para o processador. Apesar da velocidade de processadores ter aumentado de uma forma exponencial com o passar dos anos, ganhos em performance estão sendo balanceados por diversos custos cada vez maiores. Como consequência, pesquisadores e engenheiros buscam também a “paralelização” de computadores, ou seja, computadores capazes de executar mais de uma instrução por ciclo (BLELLOCH, 2015).

O hardware de um computador pode ser paralelo de diferentes maneiras, como processadores com múltiplos núcleos, SIMD (*single-instruction, multiple-data* – única instrução aplicada simultaneamente a diversos conjuntos de dados), segmentação de instruções (*pipelining*, processamento de instruções quebrado em diversos estágios, que podem ser executados simultaneamente em diferentes instruções), arquitetura VLIW (*very long instruction word*), entre outros (VISHKIN, 2010).

Para usufruir dos benefícios de paralelização de hardware, o algoritmo também deve ser implementado de uma forma paralela. O quanto que um algoritmo pode ser paralelizado depende do algoritmo, variando de algoritmos completamente não-paralelizáveis (puramente sequenciais) a algoritmos completamente paralelizáveis: Os que podem ser facilmente separados em componentes que podem ser executados de modo paralelo (HERLIHY, 2012). Frequentemente pertencem a esta classe de algoritmos os dedicados ao processamento de imagens, como o algoritmo DRLSE, no qual os diferentes pixels podem ser processados de modo puramente independente. Essa natureza paralelizável facilita especialmente a sua implementação em GPUs, um tipo de hardware com grandes quantidades de núcleos independentes.

GPUs acessíveis permitem performance computacional ordens de grandeza maiores do que processadores sequenciais, ou CPUs (ZHUGE, 2009). Por exemplo, o processador Intel i5-4460 é capaz de realizar $16 \cdot 10^6$ operações em números de ponto flutuante por segundo (FLOPS), enquanto que a placa de vídeo NVIDIA GTX 1060 6GB possibilita $4400 \cdot 10^6$ FLOPS. Essa diferença surge principalmente do alto nível de paralelização do *hardware*: CPUs modernas conseguem progredir apenas 4-12 *threads* (do inglês, linhas de execução) em paralelo, enquanto que GPUs são capazes de progredir milhares de *threads* simultaneamente (EKLUND, 2013).

GPUs foram inicialmente projetadas exclusivamente para uso em *pipelines* de computação gráfica, renderizando jogos por meio de *application programming interfaces* (APIs) como *OpenGL* ou *DirectX*. Apesar disso, desenvolvedores e pesquisadores já utilizavam GPUs para computação de propósito geral (GPGPU), manipulando o *pipeline* de computação gráfica para usos não intencionais, de modo a utilizar da grande capacidade de computação paralela do hardware (POSPICHAL, 2010; EKLUND, 2013).

A disponibilização de plataformas como *Compute Unified Device Architecture* (CUDA) (NVIDIA, 2017) ou *Open Computing Language* (OpenCL) (BOURD, 2017), facilitou grandemente o desenvolvimento de *software* GPGPU, especialmente ao remover a necessidade de compreensão e manipulação do *pipeline* de computação gráfica (POSPICHAL, 2010; EKLUND 2013).

CUDA é uma plataforma em constante desenvolvimento. Cada modelo de placa de vídeo apresenta um nível de *Compute Capability* que descreve quais das funcionalidades da plataforma o modelo suporta, e também revela alguns detalhes da arquitetura do *hardware*. A placa NVIDIA GTX 1060 6GB apresenta *Compute Capability* 6.1, por exemplo (NVIDIA, 2017).

No contexto de processamento de uma imagem bidimensional $I(x, y)$ de *lin* linhas e *col* colunas, um programa sequencial na linguagem C++ para CPU poderia aplicar o processamento $proc(k)$ ao pixel na linha *x* e coluna *y* da forma descrita na Fig. 8.

```
int main()
{
    ...
    float I[lin][col];
    float output[lin][col];

    for(int x = 0; x < lin; x++)
    {
        for(int y = 0; y < col; y++)
        {
            output[x][y] = proc(I[x][y]);
        }
    }
}
```

Figura 8 – Exemplo de processamento sequencial de uma matriz bidimensional $I[lin][col]$, com operador $proc(k)$ aplicado a cada pixel. Fonte: Daniel Coelho

A plataforma CUDA, em contraste, permite escrever funções denominadas *kernels*, identificadas pelo símbolo `__global__` e executadas em paralelo em diferentes *threads*. Um *kernel* `processPixel`, escrito em CUDA C, capaz de aplicar o processamento $proc(k)$ à *img* é indicado na Fig. 9.

```

//Kernel function
__global__ void processPixel(float* output, float* I)
{
    ...
    output[x, y] = proc(I[x, y]);
}
int main()
{
    ...
    float I[lin][col];
    float output[lin][col];
    blocks = 1;
    threadsPerBlock = 256;

    //Runs the kernel on 1 block with 256 threads
    processPixels<<<blocks, threadsPerBlock>>>(output, I);
    ...
}

```

Figura 9 – Exemplo de processamento paralelo por meio de um *kernel* *processPixel*. Uma matriz bidimensional $I[lin][col]$ tem cada pixel processado com um operador $proc(k)$. Fonte: Daniel Coelho

O paradigma de computação paralela envolve, ao invés de iterar sequencialmente sobre os pixels da imagem, fazer com que cada pixel seja processado por uma *thread* diferente. Para execução desta forma, o desenvolvedor dispara o *kernel* em um certo número de *threads*, agrupadas em conjuntos denominados blocos. Esse agrupamento de *threads* pode ser feito de maneira unidimensional, bidimensional ou tridimensional dentro de tais blocos, o que facilita determinar o pixel correspondente para cada *thread*, no caso de processamento de imagens (CHENG, 2014).

O conjunto de todos os blocos disparados é por sua vez denominado *grid*, que por sua vez também permite organizar os blocos de maneira uni-, bi- ou tridimensional.

Grids, *threads* e blocos são abstrações do processamento desejado providenciadas pela plataforma CUDA. Durante a execução, o *grid* disparado é atribuído a GPU, que decide como distribuir essas abstrações sobre seu hardware e realizar a totalidade da tarefa (CHENG, 2014; NVIDIA, 2017).

O hardware de GPUs consiste em um conjunto de núcleos, agrupados em conjuntos denominados *streaming multiprocessors* (SM). A GPU NVIDIA GTX 1060 6GB, por

exemplo, possui 10 SM, cada um com 128 núcleos. Após a atribuição de um *grid* a GPU, cada bloco é mapeado a um diferente SM, e cada *thread* mapeada a um diferente núcleo. Os núcleos executam então paralelamente e simultaneamente cada *thread* (CHENG, 2014). Todas estas definições são esquematizadas na Fig. 10.

Adicionalmente, *threads* são agrupadas em conjuntos de 32 denominados *warps*. Apesar de ser possível fazer com que *threads* de um mesmo *warp* executem instruções diferentes em um mesmo instante de tempo, para melhor desempenho computacional é essencial que todas as *threads* de um *warp* executem exatamente as mesmas instruções, geralmente em dados diferentes. Tal fato faz com que GPUs sejam classificadas como arquiteturas SIMD (CHENG, 2014).

Figura 10 – Comparação entre abstração em *software* nas estruturas *thread*, bloco e *grid* com a implementação em *hardware* das estruturas GPU, SM e núcleo. Fonte: Daniel Coelho

Na Fig. 10 é indicado que não existe necessidade de correspondência entre o *grid* executado e a implementação em hardware da GPU. Caso mais blocos sejam necessitados do que existem SM, pode ser que um ou mais blocos fiquem em espera. Do mesmo modo, se forem disparadas mais *threads* por bloco do que existem núcleos dentro de cada SM, pode ser que algumas destas *threads* sejam executadas apenas quando outras *threads* já tenham completado.

Essa separação entre abstração e implementação confere flexibilidade ao desenvolvimento, permitindo que o mesmo programa em CUDA execute em GPUs diferentes com pouca ou nenhuma mudança. A separação também explicita o fato de que o desenvolvedor não possui controle da ordem de execução dos blocos e *threads*, sendo necessário que os *kernels* desenvolvidos possam ser executados em cada thread da maneira mais independente o possível. Com isso, é de grande importância para o desempenho computacional do *kernel* que o desenvolvedor minimize a comunicação e sincronização entre *threads* e blocos (CHENG, 2014; NVIDIA, 2017).

Além de núcleos, SM possuem diversos tipos de memória e cache. Alguns destes são localizados direto no chip da GPU (*on-chip*), possuindo baixa latência e alta taxa de transferência. Outros destes tipos de armazenamento são localizados fora da GPU (*off-chip*), mas ainda sim fisicamente na placa de vídeo. Em contraste, apresentam latência relativamente mais alta e menor taxa de transferência. Uma esquematização das diversas estruturas em hardware, tipos de memória, *cache* e registros é exibida na Fig. 11, e tais componentes são descritos detalhadamente em seguida.

Cada *thread* executando em cada *core* tem acesso exclusivo e particular à um certo número de registros de 32 bits cada. A placa de vídeo NVIDIA GTX 1060 6GB, por exemplo, disponibiliza 255 registros para cada *thread*, ou 65536 registros por bloco. Sem instruções explícitas para o contrário, qualquer variável declarada internamente a um *kernel* é armazenada em um registro. Se um *kernel* usa mais registros do que o disponível, os registros em excesso são transportados para memória local (fenômeno denominado “vazamento de registros”).

Figura 11 – Esquemática de uma placa de vídeo, indicando os diversos tipos de memória e caches usados. Fonte: Daniel Coelho

Cada SM possui um segmento de memória separável em cache L1 e memória compartilhada. Ao invés de memória local ou registros, todos os núcleos de um SM têm acesso à totalidade do cache L1 e memória compartilhada, podendo inclusive usar tais memórias para comunicação e sincronização entre *threads* internas ao mesmo bloco. A única diferença entre L1 e memória compartilhada é que o primeiro é gerenciado automaticamente, enquanto que o segundo é programável, sendo controlado pelo desenvolvedor. A divisão específica da quantidade de memória entre cache L1 e memória compartilhada pode ser configurada pelo próprio software desenvolvido. Tanto o cache L1 quanto o L2 funcionam de modo análogo aos caches de mesmo nome em núcleos de CPU, servindo para armazenar valores recentemente requisitados em memórias de maior latência.

Caso o núcleo 0 necessite um certo valor da memória global, por exemplo, esse valor é replicado em L1 e L2. Se o mesmo ou outro núcleo necessitar tal valor novamente, ele é acessado direto do cache L1, que o disponibiliza com menor latência e maior taxa de transferência. Se tal valor não for utilizado, eventualmente este é despejado ao cache L1, e futuramente do cache L2, retornando ao estado inicial.

Similar em desempenho ao cache L1, cada SM possui próprias “memórias constante” e “memória somente leitura”. Tais memórias podem ser lidas e escritas pela CPU, mas apenas lidas durante a execução de *kernels*. São projetadas para otimizar o acesso a variáveis como constantes e texturas, respectivamente, que pode ser feito de modo simultâneo por múltiplas *threads* sem perda de performance (CHENG, 2014).

Texturas são abstrações de imagens uni-, bi- e tridimensionais que, apesar de serem armazenadas prioritariamente na memória global, têm segmentos copiados em cache L1 e L2 de um modo específico, com o objetivo de otimizar o acesso repetido a regiões próximas uni-, bi- ou tridimensionalmente. Caso um núcleo necessitar de um pixel do canto superior esquerdo de uma textura bidimensional, um grande bloco de diversos pixels ao redor do pixel necessitado é armazenado na memória somente leitura, acelerando operações locais em imagens, como cálculos de gradiente ou convoluções (CHENG, 2014).

É importante mencionar que L1, memória compartilhada, constante e somente leitura podem ser ausentes ou presentes em outra configuração dependendo da arquitetura da placa de vídeo. No caso da NVIDIA GTX 1060 6GB, por exemplo, o cache L1 e a memória somente leitura são combinados em um cache L1 unificado (NVIDIA, 2017).

A GPU possui apenas um outro tipo de memória *on-chip*, o cache L2. Todos os SM compartilham um único cache L2, que na placa NVIDIA GTX 1060 6GB possui capacidade de 1536 KB. Esse cache serve como intermediário ao acesso à memória global. Esta por sua vez é *off-chip* e única para a placa de vídeo inteira, servindo como armazenamento principal de dados e apresentando a maior capacidade, latência e menor taxa de transferência. Possui capacidade de 6GB na placa NVIDIA GTX 1060 6GB.

Os diferentes tipos de cache são mecanismos projetados para esconder a latência do acesso às memórias de maior capacidade como a memória global. Um SM tem a capacidade de intercalar a execução de múltiplas *threads* em um mesmo núcleo sem nenhum custo adicional, se for detectado que uma *thread* está interrompida ou aguardando a leitura de valores em memórias de grande latência. Quando a *thread* original recebe o valor necessitado, sua execução é retomada, novamente sem custo adicional para a transição (CHENG, 2014; NVIDIA, 2017).

SM só conseguem realizar tal controle em *warps* inteiros de uma única vez, porém, explicitando novamente a necessidade de que os *threads* de um mesmo *warp* operem de modo síncrono. O termo “*occupancy*” é usado para descrever a razão entre o número de *warps* ativos e o número total de *warps* executáveis em paralelo, por SM. Um *kernel* com *occupancy* de 100%, por exemplo, providencia *warps* o suficiente para que, a cada ciclo físico de processamento de seus núcleos, exista algum *warp* não interrompido disponível para execução.

4. MATERIAIS

O sistema descrito nessa dissertação foi inteiramente desenvolvido, testado e executado exclusivamente em um computador com processador AMD Ryzen 7 2700X de 8 núcleos e 3.70 Ghz, 16 GB de RAM, Windows 10 Enterprise N (64 bit) e placa de vídeo NVIDIA GTX 1060 6GB. As especificações da placa de vídeo são:

- *Compute capability* 6.1
- 10 SM com 128 núcleos CUDA cada, totalizando 1280 núcleos CUDA
- Frequência máxima de 1785 MHz
- 255 registros de 32-bit por *thread*, 65535 por bloco
- 48 KB de cache L1 combinado com memória somente leitura por SM
- 64 KB de memória constante por SM
- 96 KB de memória compartilhada por SM
- 1536 KB de cache L2
- 6 GB de memória global
- Máximo de 2048 *threads* por SM
- Máximo de 1024 *threads* por bloco
- Dimensões máximas de um bloco de (1024, 1024, 64)
- Dimensões máximas de um *grid* de (2147483647, 65535, 65535)

O ambiente de desenvolvimento envolveu o uso das seguintes bibliotecas e softwares:

- VTK v.8.0.1;
- Qt v.5.6.3;
- Qt Creator v.4.0.3;
- CUDA v.9.1;
- CMake v.3.10.1;
- Microsoft Visual C++ Compiler 14 (2015, v.140);
- Thrust para CUDA C++ v. 9.1.85.

Adicionalmente, foi utilizada a implementação de DRLSE para MATLAB produzida por Li, 2010b para comparação do desempenho do algoritmo.

O programa NVVP (NVIDIA, 2019) foi utilizado para análise da performance e comportamento dos *kernels* desenvolvidos ao longo desta dissertação.

Todas as imagens médicas utilizadas nesta dissertação foram obtidas do *The Cancer Imaging Archive (TCIA)* (CLARK, 2013). Estas são indicadas nas Fig. 12 a 16, 24, 47 a 49. Note que as imagens foram recortadas para melhor visualização.

Em cada imagem de 12 a 16 é mostrado à direita, em verde, um contorno delineado manualmente e tomado como *gold-standard*. Em vermelho, mostra-se o contorno tomado como contorno inicial no processo de treinamento. Cada conjunto imagem, *gold-standard*, e contorno inicial é denominado neste trabalho como um “contexto de segmentação”, sendo compilados na Tab. 4, junto com os correspondentes “codinomes” usados ao longo do texto.

É importante mencionar que todos os contornos traçados neste projeto, tanto contornos iniciais como contornos *gold-standard*, foram produzidos pelo autor desta dissertação e não apresentam relevância médica. A seção 6.7 busca analisar o impacto de tal escolha.

Tabela 4 – Contextos de segmentação analisados.

codinome	Figura	Modalidade	Imagem	Fonte
abdome	12	MR	116	CPTAC, 2018b
células1	13	Microscopia	113	GUPTA, 2019
cérebro1	14	MR	77	CPTAC, 2018a
cérebro2	15	MR	73	CPTAC, 2018a
pélvis1	16	CT	588	ROTH, 2015

Figura 12 – Imagem 116, série 7-Ax 3D DualEcho BH-37647, estudo 03-24-2003-MR ABDOMEN NONENHANCED ENHANCEDAB-13326, paciente C3L-00598. Fonte: CPTAC, 2018b. Originalmente 512x512 pixels. Modalidade MR. Imagem superior: sem segmentações. Imagem inferior: com contorno de referência em verde, e contorno inicial em vermelho.

Figura 13 – Imagem 113, estudo MiMM_SBILab. Fonte: GUPTA, 2019. Originalmente 2560x1920 pixels. Imagem de microscopia. À esquerda sem segmentações, à direita com contornos de referência em verde, e contorno inicial em vermelho.

Figura 14 – Imagem 77, série 801-sT1W3DTFEAX POST-99314, estudo 04-23-2000-MR BRAIN WOW CONTRAST-18837, paciente C3L-00629. Fonte: CPTAC, 2018a.

Originalmente 512x512 pixels. Modalidade MR. À esquerda sem segmentações, à direita com contorno de referência em verde, e contorno inicial em vermelho.

Figura 15 – Imagem 73, série 701-sT1W3DTFEAX PRE-30908, estudo 04-23-2000-MR BRAIN WOW CONTRAST-18837, paciente C3L-00629. Fonte: CPTAC, 2018a. Originalmente 528x528 pixels. Modalidade MR. À esquerda sem segmentações, à direita com contorno de referência em verde, e contorno inicial em vermelho.

Figura 16 – Imagem 588, série abdominallymphnodes-03108, estudo 09-14-2014-ABDLYMPH005-abdominallymphnodes-90939, paciente ABD_LYMPH_005. Fonte: ROTH, 2015. Originalmente 512x512 pixels. Modalidade CT. Acima sem segmentações, abaixo com contornos de referência em verde, e contorno inicial em vermelho.

5. METODOLOGIA

Em linhas gerais, o sistema desenvolvido funciona em dois estágios, aprendizado e aplicação (Fig. 17).

Figura 17 – Esquematização do sistema desenvolvido. Fonte: Daniel Coelho

No estágio de aprendizado, o usuário produz manualmente uma segmentação de referência, que seja representativa da segmentação final que deseja obter (em verde nas Fig. 12 a 16). Em seguida, este produz uma segmentação inicial “grosseira”, representativa de uma rápida segmentação que faria durante o estágio de aplicação (em vermelho nas Fig. 12 a 16). Ambas as segmentações são providenciadas ao estágio de treinamento. Neste estágio, algoritmos genéticos são usados para determinar o conjunto de parâmetros que faz com que a segmentação produzida pelo algoritmo DRLSE, a partir da segmentação inicial, mais se aproxime da segmentação de referência. Isso é feito usando métodos de avaliação de segmentação para o cálculo de *fitness* de cada

indivíduo, cada um representando por um conjunto de parâmetros. Quando um *fitness* alto o suficiente é observado, ou um número fixo de gerações é executado, o melhor conjunto de parâmetros é então retornado ao usuário.

Em seguida, no estágio de aplicação, o usuário utiliza o conjunto de parâmetros obtido e a mesma segmentação grosseira como entrada para uma simples execução do algoritmo DRLSE. O algoritmo evolui o contorno da segmentação grosseira, com o intuito de que se aproxime o máximo o possível da segmentação de referência, obtendo a segmentação final. Com tal conjunto de parâmetros em mãos, porém, é possível utilizar o mesmo sistema para rapidamente segmentar imagens visualmente similares, a partir de segmentações grosseiras análogas à primeira.

5.1. Interface gráfica

Foi desenvolvida uma interface gráfica com as bibliotecas VTK e Qt para permitir a delimitação manual dos contornos inicial e de referência. A biblioteca VTK foi escolhida pois permite o fácil uso de imagens em diversos formatos, e possui boa integração com a biblioteca Qt, frequentemente usada para desenvolvimento de interfaces gráficas em C++. A Fig. 18 mostra uma vista da versão atual da interface implementada.

A Fig. 18 indica como a interface gráfica do software permite facilmente salvar e carregar conjuntos de parâmetros (a esquerda), contornos (a direita, inferior) e imagens (a direita, superior). Com o menu “DRLSE”, no topo, é possível escolher entre treinar um conjunto de parâmetros a partir de contornos inicial e de referência (Fig. 19A), ou usar um conjunto de parâmetros e um contorno inicial para realizar uma simples segmentação com o algoritmo DRLSE (Fig. 19B).

Figura 18 – Interface gráfica desenvolvida, exibindo a imagem descrita na Fig. 13, em processo de delimitação de um contorno, em verde. Fonte: Daniel Coelho

Figura 19 – Diferentes opções para o uso do algoritmo DRLSE. A: Contornos iniciais, e de referência (*gold-standard*) são usados para treinamento, originando um conjunto de parâmetros. B: Um conjunto de parâmetros e um contorno inicial são usados para uma segmentação simples. Fonte: Daniel Coelho

5.2. Pré-processamento dos dados

Tanto o contorno inicial quanto o contorno de referência são inicialmente dados poligonais, contendo apenas vértices e segmentos de reta. Estes devem ser rasterizados em imagens para serem processados pelo algoritmo DRLSE e os algoritmos de avaliação de segmentação. Para isso, são utilizados o filtro *vtkPolyDataToImageStencil* e a classe *vtkImageStencil* da biblioteca VTK, que geram imagens das mesmas dimensões que a imagem à segmentar, mas cujos pixels apresentam 0 se se encontram fora da região delimitada pelo contorno, e 1 se se encontram dentro da mesma (Fig. 20).

Figura 20 – Rasterização de um contorno poligonal (vermelho) em uma segmentação binária, onde pixels em preto descrevem áreas fora do contorno, e pixels em cinza descrevem áreas dentro do contorno. Fonte: Daniel Coelho

O contorno de referência rasterizado é deixado no intervalo $[0, 1]$ e será usado apenas nos algoritmos de avaliação de segmentação. O contorno inicial rasterizado, porém, tem seu intervalo $[0, 1]$ mapeado linearmente ao intervalo $[-c_0, c_0]$, sendo c_0 uma constante descrita em LI, 2010a. Em termos práticos, tal constante descreve a

largura da região regularizada pelo algoritmo DRLSE ao redor do conjunto nível zero (Fig. 21, a direita), e LI, 2010a recomenda o uso de $c_0 \geq 1$. Note que a regularização faz com que a magnitude do gradiente se aproxime de 1 em tal região, de modo que sua largura se aproxima de $2c_0$. Neste trabalho, observou-se bons resultados com $c_0 = 9$.

Figura 21 – Progresso no processamento de uma seção transversal de uma segmentação. À esquerda: Contorno rasterizado. No centro: Contorno rasterizado mapeado ao intervalo $[-c_0, c_0]$ (ϕ inicial). À direita: ϕ após algumas iterações. Fonte: Daniel Coelho

A imagem a segmentar também é pré-processada: Os valores de intensidade são sempre mapeados ao intervalo $[0,0, 1000,0]$, tanto para treinamento como para aplicação. A imagem indicadora de bordas $g(x,y)$ é calculada a partir de tal versão normalizada da imagem.

Em termos de implementação, note que números de ponto flutuante de 32 bits foram usados para representação de tais valores de intensidade mapeados, de modo que o mapeamento em si não acarreta em perda de precisão.

O intervalo específico de $[0,0, 1000,0]$ foi escolhido empiricamente, e proporciona bons resultados para a formulação de $g(x,y)$. Para compreender a necessidade de tal normalização de modo geral, observe a formulação de $g(x,y)$, replicada abaixo na equação 24: Uma imagem com pixels de intensidades no intervalo $[0,0, 5,0]$ apresenta $g(x,y)$ com intensidades no intervalo $[0,1379, 1,000]$. A mesma imagem com intensidades multiplicadas por 10, ou seja, com intensidades no intervalo $[0,0, 50,0]$, apresenta $g(x,y)$ com intensidades no intervalo $[0,0016, 1,000]$. A magnitude de $g(x,y)$ modula o quanto que o conjunto nível zero de ϕ é atraído pelas regiões de $g(x,y)$, de modo que sem tal normalização, a evolução de ϕ seria diferente para as duas imagens.

Tal comportamento não é desejado, especialmente quando se usa imagens de modalidades diferentes.

5.3. Implementação de DRLSE em CUDA

A imagem indicadora de bordas $g(x, y)$, cuja formulação é replicada na equação 24, pode ser calculada por meio do algoritmo 1, executado na CPU. O passo 3 de tal algoritmo envolve a execução do *kernel* *EdgeIndicatorKernel* na GPU, que é descrito no algoritmo 2.

$$g(x, y) = \frac{1}{1 + |\nabla G_{\sigma} * I(x, y)|^2} \quad (24)$$

Note que *imagem_borda_grad* possui as mesmas dimensões que *imagem*, mas possui 2 canais – descrevendo o gradiente nas direções x e y .

Algoritmo 1 – Imagem indicadora de bordas $g(x, y)$, CPU

Entradas: *núcleo_gaussiano_15x15*, *imagem*

Procedimento:

1. Execute *ConvolutionKernel* com *núcleo_gaussiano_15x15* e *imagem*, gerando *imagem_temp*
2. Defina *imagem_borda* e *imagem_borda_grad*, ambas com as mesmas dimensões que *imagem*
3. Execute *EdgeIndicatorKernel* com *imagem_temp* e *imagem_borda*, alterando *imagem_borda*
4. Execute *GradientKernel* com *imagem_borda* e *imagem_borda_grad*, alterando *imagem_borda_grad*
5. Retorne *imagem_borda* e *imagem_borda_grad*

Saídas: *imagem_borda* e *imagem_borda_grad*

Após a construção da imagem indicadora de bordas $g(x, y)$, o principal processo de iteração do DRLSE é executado na CPU. Esse processo consiste em repetidamente executar três *kernels* até o número alvo de iterações, e é descrito no algoritmo 3. A principal função de tal algoritmo é preparar, para a iteração atual, as imagens intermediárias *imagem_phi_grad* e *imagem_laplace*, e em seguida executar o algoritmo *LevelSetKernel*, descrito no algoritmo 4, consumindo tais imagens. Note que tais imagens são calculadas com os *kernels* *NormalizedGradientKernel* e *LaplaceKernel*,

respectivamente. Estes são omitidos aqui, mas tratam-se de simples cálculos de gradiente por diferenças centrais, e convolução com núcleo de Laplace, operações também discutidas na seção 5.4.

Algoritmo 2 – *EdgeIndicatorKernel*, GPU

Entradas: *imagem_temp*, *imagem_borda*

Procedimento:

1. Adquirir a posição $[x, y]$ do pixel atual;
2. Defina $plusX = imagem_temp[x + 1, y]$;
3. Defina $minX = imagem_temp[x - 1, y]$;
4. Defina $plusY = imagem_temp[x, y + 1]$;
5. Defina $minY = imagem_temp[x, y - 1]$;
6. Defina $gradX = plusX - minX$;
7. Defina $gradY = plusY - minY$;
8. Defina $g = 1.0f / (1.0f + 0.25f * (gradX * gradX + gradY * gradY))$;
9. Escreva $imagem_borda[x, y] = g$;

Saídas: Nenhuma

É importante notar que *imagem_phi_grad* possui quatro canais. Os dois primeiros descrevem o gradiente de *phi* naquele pixel, nas direções x e y . O terceiro e quarto canal correspondem à *xnorm* e *ynorm*, e descrevem o mesmo gradiente normalizado de modo que sua magnitude seja igual a 1. Agrupar tais dados desta forma proporciona melhor performance, já que tais valores podem ser lidos de uma única vez.

Algoritmo 3 – Iteração de DRLSE, CPU

Entradas: *imagem*, *imagem_phi*, *imagem_borda*, *imagem_borda_grad*, *mu*, *lambda*, *alpha*

Procedimento:

1. Defina *imagem_phi_grad* e *imagem_laplace*, ambos com as mesmas dimensões de *imagem*
2. Para 1:1000
3. Execute *GradientNormalizedKernel* com *imagem_phi* e *imagem_phi_grad*, modificando *imagem_phi_grad*
4. Execute *LaplaceKernel* com *imagem_phi* e *imagem_laplace*, modificando *imagem_laplace*
5. Execute *LevelSetKernel* com *mu*, *lambda*, *alpha*, *imagem_phi*, *imagem_borda*, *imagem_borda_grad*, *imagem_phi_grad* e *imagem_laplace*, modificando *imagem_phi*
6. Retorne *imagem_phi*

Saídas: *imagem_phi*

O passo 5 do algoritmo 3 envolve a execução do *LevelSetKernel*, descrito no algoritmo 4. Este é o principal *kernel* da implementação, e corresponde à equação de

iteração da formulação do algoritmo DRLSE, replicada na equação 25. Este *kernel* é maior que o normal para minimizar os custos adicionais de despachar *kernels*.

$$\phi_{x,y}^{k+1} = \phi_{x,y}^k + \tau \cdot \left[\begin{array}{c} \mu \cdot \operatorname{div} \left(\frac{p'(|\nabla \phi_{x,y}^k|)}{|\nabla \phi_{x,y}^k|} \nabla \phi_{x,y}^k \right) + \\ \lambda \cdot \delta_\varepsilon(\phi_{x,y}^k) \cdot \operatorname{div} \left(g(x,y) \cdot \frac{\nabla \phi_{x,y}^k}{|\nabla \phi_{x,y}^k|} \right) + \\ \alpha \cdot g(x,y) \cdot \delta_\varepsilon(\phi_{x,y}^k) \end{array} \right] \quad (25)$$

O procedimento *DistRegPre* usado na linha 11 do algoritmo 4, e descrito no algoritmo 5 corresponde à parte do cálculo do termo regularizador de distância, ponderado pela constante μ na formulação replicada na equação 25. O algoritmo 5 recebe um vetor com quatro canais e retorna um vetor com dois canais. Note que tal algoritmo repetidamente usa o fato de que *falso* == 0 e *verdadeiro* == 1, que permite o comportamento de seleção dentre duas alternativas (ponderadas por a e b , no passo 4), sem o custo adicional atrelado ao desvio de fluxo de controle e potencial divergência entre *threads* de um mesmo *warp*.

Algoritmo 4 – *LevelSetKernel*, GPU

Entradas: *mu*, *lambda*, *alpha*, *imagem_phi*, *imagem_borda*, *imagem_borda_grad*, *imagem_phi_grad*, *imagem_laplace*

Procedimento:

1. Adquirir a posição $[x, y]$ do pixel atual;
2. Defina $\phi = \text{imagem_phi}[x, y]$;
3. Defina $\phi_grad = \text{imagem_phi_grad}[x, y]$;
4. Defina $borda = \text{imagem_borda}[x, y]$;
5. Defina $borda_grad = \text{imagem_borda_grad}[x, y]$;
6. Defina $plusX = \text{imagem_phi_grad}[x + 1, y]$;
7. Defina $minX = \text{imagem_phi_grad}[x - 1, y]$;
8. Defina $plusY = \text{imagem_phi_grad}[x, y + 1]$;
9. Defina $minY = \text{imagem_phi_grad}[x, y - 1]$;
10. Defina $curvatura = 0.5f * (plusX.xnorm - minX.xnorm + plusY.ynorm - minY.ynorm)$;
11. Defina $dist_reg = \text{imagem_laplace}[x, y] + 0.5f * \text{DistRegPre}(plusX).x - \text{DistRegPre}(minX).x + \text{DistRegPre}(plusY).y - \text{DistRegPre}(minY).y$;
12. Defina $delta = \text{DiracDelta}(\phi)$;
13. Defina $incremento = \tau * (\mu * dist_reg + \lambda * delta * (borda_grad.x * \phi_grad.xnorm + borda_grad.y * \phi_grad.ynorm + borda * curvatura) + \alpha * delta * borda)$;
14. Escreva $\phi[x, y] = \phi + incremento$;

Saídas: Nenhuma

O algoritmo *DistRegPre* em 5 é o que mais se distancia de uma correspondência 1-para-1 com a formulação do algoritmo DRLSE, replicada na equação 25. Tal implementação do algoritmo 5 corresponde, porém, quase que exatamente ao procedimento análogo na implementação de referência de Li, 2010b, com leve modificações em prol de melhor performance.

Algoritmo 5 – *DistRegPre*, GPU

Entradas: *phi_grad*

Procedimento:

1. Defina $mag = \sqrt{phi_grad.x^2 + phi_grad.y^2}$;
2. Defina $a = (mag > 0) \ \&\& \ (mag \leq 1.0)$;
3. Defina $b = (mag > 1.0)$;
4. Defina $ps = a \cdot \frac{\sin(2\pi \cdot mag)}{2\pi} + b \cdot (mag - 1.0)$;
5. Defina $dps = ((ps \neq 0.0) \cdot ps + (ps == 0.0)) / ((mag \neq 0.0) \cdot mag + (mag == 0)) - 1.0$;
6. Defina $res = [dps \cdot phi_grad.x, \ dps \cdot phi_grad.y]$;
7. Retorne res ;

Saídas: res

Na linha 12 do algoritmo 4, o procedimento *DiracDelta* é executado. Este procedimento é descrito pelo algoritmo 6, e trata-se da implementação da equação 26, replicada da formulação do algoritmo DRLSE. Note que ε ou *epsilon*, valor que descreve o formato da aproximação da função Delta de Dirac, foi sempre mantido no valor de 1,5, sugerido em Li, 2010a.

$$\delta_\varepsilon(s) = \begin{cases} \frac{1}{2\varepsilon} \left[1 + \cos\left(\frac{\pi x}{\varepsilon}\right) \right], & |x| \leq \varepsilon \\ 0, & |x| > \varepsilon \end{cases} \quad (26)$$

Algoritmo 6 – *DiracDelta*, GPU

Entradas: x, ϵ

Procedimento:

1. Defina $f = (0.5/\epsilon) \cdot (1.0 + \cos(\pi \cdot x/\epsilon))$;
2. Defina $b = (x \leq \epsilon) \ \&\& \ (x \geq -\epsilon)$;
3. Defina $res = f \cdot b$;
4. Retorne res ;

Saídas: res

5.4. Comparações entre implementações de DRLSE

Para enriquecer as comparações em performance, o algoritmo DRLSE foi também implementado em C++ de modo puramente sequencial.

Com o objetivo de atingir um equilíbrio entre performance e concordância entre os resultados, os operadores necessários foram implementados, tanto em CUDA quanto em C++, de modo similar à referência do software MATLAB – linguagem escolhida para a implementação de referência do algoritmo DRLSE (LI, 2010b).

Temos por exemplo que gradientes foram implementados de acordo com diferenças centrais:

$$grad[x] = (image[x + 1] - image[x - 1]) \cdot 0,5 \quad (27)$$

A segunda derivada foi implementada via convolução com um núcleo laplaciano discreto (exibido na equação 28), equivalendo ao procedimento $4 \cdot del2(image)$ realizado em MATLAB.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (28)$$

Com o intuito de eliminar a influência de bibliotecas de leitura de imagens DICOM e processamento de imagens, todas as imagens utilizadas neste trabalho foram exportadas diretamente como vetores de números de ponto flutuante de 32 bits. Desse modo, as implementações em C++, CUDA e MATLAB utilizam exatamente os mesmos dados de entrada.

5.5. Otimização dos parâmetros do algoritmo DRLSE

Os parâmetros controláveis pelo usuário na interface do software desenvolvido são indicados na Tab. 5.

Tabela 5 – Parâmetros controlados pelo usuário.

Parâmetro	Intervalo razoável	Interpretação
μ , ou <i>mu</i>	(0,30]	Peso do termo de regularização de distância do DRLSE
λ ou <i>lambda</i>	(0,30]	Peso do termo de comprimento do DRLSE
α ou <i>alpha</i>	[−30,30]	Peso do termo de área do DRLSE
σ ou <i>sigma</i>	[0,30]	Desvio padrão do núcleo gaussiano convolvido com $g(x,y)$
τ ou <i>timestep</i>	(0,1.25)	Intervalo de tempo entre iterações (ou peso do incremento de ϕ)
i_l ou <i>late iter</i>	[0,1]	Fração das iterações executada em modo inicial
μ_l ou <i>late mu</i>	(0,30]	Valor tardio para μ
λ_l ou <i>late lambda</i>	(0,30]	Valor tardio para λ
α_l ou <i>late alpha</i>	[−30,30]	Valor tardio para α
τ_l ou <i>late timestep</i>	(0,1.25)	Valor tardio para τ

Note que apesar de todos tais valores serem manualmente configuráveis pelo usuário para a execução do algoritmo DRLSE em modo de aplicação, para o modo de treinamento temos que apenas os parâmetros $\lambda, \alpha, \sigma, \tau, i_l, \lambda_l, \alpha_l$ e τ_l são considerados para otimização, ou seja, μ e μ_l são fixados em algum valor qualquer. Isso é consequência de que os três parâmetros principais do algoritmo DRLSE (μ, λ e α) são apenas diferentes pesos para os três termos da equação otimizada: É sempre possível normalizar os três termos por um deles (equação 29), de modo que um dos parâmetros (no caso μ) pode ser fixado. No software desenvolvido, 0.2 foi arbitrariamente escolhido, de modo que todos os conjuntos de parâmetros gerados pelo processo de treinamento apresentam $\mu = \mu_l = 0,2$.

$$\mu' = \frac{\mu}{\mu} = 1, \quad \lambda' = \frac{\lambda}{\mu}, \quad \alpha' = \frac{\alpha}{\mu} \quad (29)$$

Os intervalos razoáveis descritos para tais parâmetros na Tab. 5 são combinação das restrições da formulação do DRLSE de LI, 2010a ($\mu > 0, \lambda > 0$), com intervalos razoáveis observados empiricamente para as imagens trabalhadas. Adicionalmente, a restrição de Courant-Friedrichs-Lewy $\mu \cdot \tau < \frac{1}{4}$ e a fixação de $\mu = \mu_l = 0,2$ limitam τ e τ_l para o intervalo (0, 1,25).

Em uma execução do algoritmo DRLSE com T iterações, temos que os parâmetros μ, λ, α e τ são utilizados nas iterações pertencentes ao intervalo $[1, \text{round}(T \cdot i_l)]$, denominado modo inicial. Nas iterações do intervalo $[\text{round}(T \cdot i_l) + 1, T]$, inicia-se o “modo tardio”, e os parâmetros μ, λ, α e τ são trocados por $\mu_l, \lambda_l, \alpha_l$ e τ_l . Tal mecanismo

permite, por exemplo, que um único conjunto de parâmetros possua α com módulo alto e $\alpha_l = 0$, de modo que o conjunto nível zero se expanda ou contraia rapidamente em modo inicial, mas não observe tal tendência no modo tardio, permitindo um refinamento final à borda alvo. Obviamente, um indivíduo pode possuir $l_i = 1,0$, significando que μ_l , λ_l , α_l e τ_l não são utilizados, ou $l_i = 0,0$, significando que apenas λ_l , α_l e τ_l são utilizados. Empiricamente nota-se que isso acontece em boa parte dos treinamentos, mas não sempre.

5.6. Implementação dos algoritmos genéticos

Todos os operadores dos algoritmos genéticos foram implementados puramente em C++, sem uso de GPU. Isso é consequência de que foi observado ao longo da implementação e evolução do sistema que a iteração do algoritmo DRLSE era suficiente para saturar completamente a placa de vídeo, e será discutido mais profundamente na seção 6.3.1.

A Tab. 6 indica os diversos operadores implementados, bem como os parâmetros de cada estágio.

Tabela 6 – Operadores genéticos implementados.

Operador	Referência	Parâmetros
Operadores de seleção		
Torneios	GOLDBERG, 1991A	Tamanho do torneio T_o
Ranking linear	BAKER, 1985	-
Ranking exponencial	BLICKLE, 1995	Base do expoente b
Amostragem universal estocástica	BAKER, 1987	-
Truncamento	MÜHLENBEIN e SCHLIERKAMP-VOOSEN, 1993	Fração de truncamento T_r
Operadores de recombinação		
Dois pontos	WHITLEY, 1994	P_c
Uniforme	WHITLEY, 1994	P_c
Aritmético	MICHALEWICZ, 1996	P_c
BLX- α	ESCHELMAN e SHAFFER, 1993	P_c, α_{BLX}
SBX	DEB, 1995	P_c, η_{SBX}
Operadores de mutação		
Uniforme	MICHALEWICZ, 1996	P_M
Gaussiana	HINTERDING, 1995	P_M, σ_m

Ao longo do treinamento, a maioria do tempo é dedicada ao cálculo dos valores de *fitness* de cada indivíduo. Para cada um, o algoritmo DRLSE é aplicado ao contorno inicial usando os parâmetros do indivíduo, e a segmentação resultante é avaliada (discutida na seção seguinte).

Os algoritmos genéticos implementados podem utilizar de elitismo, preservando uma fração dos melhores indivíduos de uma geração na geração seguinte. Especialmente com isso em mente, foi implementado um cache, mapeando todos os indivíduos observados aos seus correspondentes valores calculados de *fitness*. Desse modo, toda vez que se avalia o *fitness* de um indivíduo, o cache é checado primeiro, potencialmente eliminando cálculos redundantes. Se o valor não se encontra no cache, a segmentação com algoritmo DRLSE é realizada e avaliada, e o valor calculado de *fitness* é adicionado ao cache. Tal cache é implementado combinando-se o *hash* dos diferentes parâmetros otimizados ($\lambda, \alpha, \sigma, \tau, i_l, \lambda_l, \alpha_l$ e τ_l) para o indivíduo.

5.7. Avaliação de segmentação

A avaliação da segmentação obtida ao se aplicar o algoritmo DRLSE a um conjunto de parâmetros e contorno inicial é feita comparando-se o segmento resultante com a segmentação de referência, *gold-standard*. Para maximização de performance, a computação necessária para tais comparações foi realizada de modo paralelo em CUDA o máximo o possível. Apesar disso, alguns estágios de tal processo não são paralelizáveis ou apresentam melhor performance em CPU por outros motivos, e foram então realizados em CPU.

Adicionalmente, quando possível tentou-se minimizar o número de vezes que memória é copiada de CPU à GPU ou de GPU à CPU. Tais transmissões requerem sincronização entre CPU e GPU, e apresentam alto custo em termos de performance. Com isso em mente, os métodos de avaliação de segmentação avaliam simultaneamente as segmentações de todos os indivíduos, e retornam todos os resultados em formato de vetor, uma única vez.

5.7.1. Implementação do coeficiente de Dice

A formulação do coeficiente de Dice é replicada na equação 30. Neste, note que S_a^1 refere-se ao conjunto de pixels pertencentes ao segmento produzido na segmentação S , e S_r^1 refere-se ao segmento produzido pela segmentação de referência. Aqui, o operador $|\cdot|$ refere-se à cardinalidade de tal segmento, ou seja, o número de pixels contidos neste.

$$DICE = \frac{2|S_a^1 \cap S_r^1|}{|S_a^1| + |S_r^1|} \quad (30)$$

O algoritmo 7 descreve a implementação completa do coeficiente de Dice. Note que a área do segmento de referência, gs_area , é calculada previamente uma única vez, já que é constante durante a evolução de cada segmentação.

Algoritmo 7 – *CoefDICE*, CPU

Entradas: *imagem_phi*, *imagem_gs_b*, *gs_area*

Procedimento:

1. Defina *imagem_phi_b* e *imagem_inter_b*, ambos com as mesmas dimensões de *imagem_phi*;
2. Execute *CoefDICEKernel* com *imagem_phi*, *imagem_gs_b*, *imagem_phi_b*, *imagem_inter_b*, modificando *imagem_phi_b* e *imagem_inter_b*;
3. Execute *thrust :: reduce_by_key* com *imagem_phi_b* e *thrust :: plus*, calculando *phi_area*;
4. Execute *thrust :: reduce_by_key* com *imagem_inter_b* e *thrust :: plus*, calculando *inter_area*;
5. Defina $DICE = 2.0 \cdot inter_area / (phi_area + gs_area)$;
6. Retorne *DICE*;

Saídas: *DICE*

A etapa 2 do algoritmo 7 envolve a execução do *kernel CoefDICEKernel* na GPU. Este é descrito no algoritmo 8. Com o intuito de minimizar o custo adicional de despachar *kernels* adicionais, *CoefDICEKernel* realiza duas tarefas: A binarização de *imagem_phi*, preenchendo *imagem_phi_b*, e o cálculo da intersecção de *imagem_phi_b* e *imagem_gs_b*, preenchendo *imagem_inter_b*. Note que na etapa 4 do algoritmo 8, foi escolhido binarizar $\phi \in \mathbb{R}$ de modo que o intervalo $[0, +\infty)$ é mapeado ao valor 1, e $(-\infty, 0)$ mapeado ao valor 0.

Algoritmo 8 – CoefDICEKernel, GPU

Entradas: *imagem_phi*, *imagem_gs_b*, *imagem_phi_b*, *imagem_inter_b*
Procedimento:

1. Adquirir a posição $[x, y]$ do pixel atual;
2. Defina $\phi = \text{imagem_phi}[x, y]$;
3. Defina $gs_b = \text{imagem_gs_b}[x, y]$;
4. Defina $\phi_b = \phi \geq 0.0$;
5. Escreva $\text{imagem_phi_b}[x, y] = \phi_b$;
6. Escreva $\text{imagem_inter_b}[x, y] = gs_b * \phi_b$;

Saídas: Nenhuma

O cálculo de cardinalidade (ou área) de um segmento em uma imagem não é uma tarefa facilmente paralelizável. Felizmente, a biblioteca *Thrust* (BELL, 2019) proporciona *kernels* prontos para a realização de algumas operações comuns, como reduções, que são úteis neste contexto. Note que para a imagem binarizada, um pixel dentro do segmento possui valor 1, enquanto que um pixel fora do segmento possui valor 0. Deste modo, o cálculo de área consiste na simples soma de todos os pixels da imagem, ou seja, na *redução* de uma imagem a um único valor, por meio da operação soma (*thrust :: plus*).

Nas etapas 3 e 4 do algoritmo 7, foi utilizada a função *reduce_by_key*, que nos permite realizar diversas reduções simultâneas. Tal operação permite a avaliação da segmentação de todos os indivíduos de modo paralelo, retornando à CPU um vetor com todos os resultados de uma única vez, ao invés de pagar o alto custo de comunicação CPU-GPU cada vez para cada indivíduo. Note então que por simplicidade, o algoritmo 7 trata *phi_area*, *inter_area* e *DICE* como se fossem simples variáveis, mas na verdade estes descrevem vetores, e a operação na etapa 5 é feita membro-a-membro.

5.7.2. Implementação da Distância de Hausdorff

A formulação da Distância de Hausdorff HD entre dois conjuntos de pontos A e B é replicada abaixo na equação 31.

$$HD(A, B) = \max(h(A, B), h(B, A)) \quad (31)$$

Sendo $h(A, B)$ denominada distância de Hausdorff direcionada, e é dada pela equação 32, onde a e b indicam pontos de A e B , respectivamente.

$$h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\| \quad (32)$$

A mesma formulação pode ser usada caso A e B descrevam segmentos, e a e b *spels* pertencentes a tais segmentos.

Por motivos de performance, apenas a distância de Hausdorff direcionada é utilizada neste trabalho, descrevendo sempre a distância do segmento presente em ϕ em relação ao segmento presente na imagem de referência GS , ou seja:

$$HD(A, B) = HD(\phi, GS) = h(\phi, GS) = \max_{a \in \phi} \min_{b \in GS} \|a - b\| \quad (33)$$

A principal consequência negativa de usar apenas a distância de Hausdorff direcionada é que está é assimétrica: concavidades excessivas nos contornos podem gerar valores incorretos para a métrica, e tais valores são diferentes caso tal concavidade se encontre no resultado da segmentação ou no *gold-standard*. Tal consequência é analisada na seção 6.1, e é parcialmente compensada pelo fato de que, como será discutido na seção seguinte, a métrica completa usada para avaliação de segmentações envolve a multiplicação da distância de Hausdorff com o coeficiente de Dice. No caso geral, segmentações em que os segmentos comparados diferem o suficiente para que tal consequência apresente efeito significativo teriam intersecções tão baixas que a influência do coeficiente de Dice faz com que o valor da métrica total tenda a zero, independentemente do valor de HD .

O principal benefício de restringir à distância de Hausdorff apenas à distância direcionada é que uma transformada de distância de GS , *imagem_gs_dist* pode ser calculada uma única vez e usada em todas as avaliações, reduzindo significativamente parte da computação necessária à simples (e paralelizável) operação de consultar *imagem_gs_dist*. Note que cada pixel p em *imagem_gs_dist* descreve a mínima distância euclidiana, em pixels, de p a qualquer pixel do segmento descrito em GS . Um exemplo é indicado na Fig. 22.

Figura 22 – Imagem em preto e branco (à esquerda), e sua transformada de distância (à direita), indicando a mínima distância ao pixel branco mais próximo. Valores baixos de distância são representados com tons escuros de cinza. Fonte: Daniel Coelho

O algoritmo 9 descreve o procedimento em CPU usado para o cálculo de HD . A etapa 2 do algoritmo 9 envolve o uso do *kernel ImageOfMins*, descrito no algoritmo 10, que calcula para cada pixel do conjunto nível zero de *imagem_phi*, a distância mínima à qualquer pixel do segmento de *imagem_gs_dist*. Tais valores são escritos nos pixels correspondentes de *imagem_mins*. Pixels de *imagem_mins* que não se referem a pixels do conjunto nível zero de *imagem_phi* recebem valor $-1,0$.

Na etapa 3 do algoritmo 9, *thrust :: reduce_by_key* é utilizado para reduzir *imagem_mins* a um único valor HD , realizando repetidamente a operação *thrust :: max* entre conjuntos de pixels de tal imagem.

Algoritmo 9 – Distância de Hausdorff, CPU

Entradas: *imagem_phi*, *imagem_gs_dist*, *dist_max*

Procedimento:

1. Defina *imagem_mins*, com as mesmas dimensões de *imagem_phi*;
2. Execute *ImageOfMins* com *imagem_phi*, *imagem_gs_dist* e *imagem_mins* modificando *imagem_mins*;
3. Execute *thrust :: reduce_by_key* com *imagem_mins* e *thrust :: max*, calculando HD ;
4. Se $HD < 0,0$, faça $HD = dist_max$;
5. Defina $HDa = (dist_max - HD)/dist_max$;
6. Retorne HDa ;

Saídas: HDa

Algoritmo 10 – *ImageOfMins*, GPU

Entradas: *imagem_phi*, *imagem_gs_dist* e *imagem_mins*

Procedimento:

1. Adquira a posição $[x, y]$ do pixel atual;
2. Defina $\phi = \text{imagem_phi}[x, y]$;
3. Defina $gs_dist = \text{imagem_gs_dist}[x, y]$;
4. Defina $zls = \phi > -0.51 \ \&\& \ \phi < 0.51$;
5. Se zls :
6. Escreva $\text{imagem_mins}[x, y] = gs_dist$;
7. Senão:
8. Escreva $\text{imagem_mins}[x, y] = -1.0$;

Saídas: Nenhuma

Observe que na etapa 4 do algoritmo 9, o teste $HD < 0,0$ é realizado. Isso é consequência de que o *kernel ImageOfMins* retorna $-1,0$ nos casos em que o pixel não descreve o conjunto nível zero. Caso *imagem_phi* contenha exclusivamente tais pixels (o que pode ocorrer em segmentações degeneradas e inicializações), a redução de *imagem_mins*, HD , também pode apresentar valor $-1,0$. Nesse caso, o teste da etapa 4 faz com que seja retornado $HDa = 0$.

Finalmente, com o intuito de mapear HD a um intervalo similar ao dos valores do coeficiente de DICE, ou seja, $[0, 1]$ onde 1 descreve o melhor valor, $dist_max$ é previamente calculado de acordo com a equação 34, sendo $altura$ e $largura$ as dimensões de *imagem_phi*, e dado como entrada ao algoritmo 9. Tal valor é usado na simples normalização descrita na etapa 5 de tal algoritmo, calculando a distância de Hausdorff adaptada HDa .

$$dist_max = \sqrt{altura^2 + largura^2} \quad (34)$$

Note na etapa 4 do *kernel ImageOfMins* que a variável zls descreve se o pixel x, y pertence ao conjunto nível zero de *imagem_phi*. Seu valor é obtido testando se o valor de ϕ se encontra no intervalo $(-0,51, 0,51)$. Tal teste é baseado em três fatos:

- A. A influência do efeito do termo de regularização de distância do algoritmo DRLSE faz com que, na proximidade do conjunto nível zero, intensidades de pixels vizinhos tendam a variar com magnitude de gradiente próximo de 1;

- B. *imagem_phi* descreve uma imagem bidimensional discreta, efetivamente uma quantização de $\phi: \mathbb{R}^2 \rightarrow \mathbb{R}$ nos centros dos pixels, de modo que exceto no caso que o conjunto nível zero atravesse o exato centro do pixel, um pixel que represente o conjunto nível zero pode apresentar valor em $[-0,5, 0,5]$;
- C. Ao longo da evolução de *phi*, a magnitude do gradiente de *imagem_phi* pode temporariamente e localmente desviar do valor 1.

Foi observado experimentalmente que o intervalo $(-0,51, 0,51)$ é razoável para determinar de modo consistente o pixel que descreve o conjunto nível zero, ou seja, o pixel com menor valor em módulo.

5.7.3. Implementação da métrica combinada Q_S

Com o intuito de se obter uma métrica genérica, capaz de produzir bons resultados independentemente do contexto e tipo de segmentação escolhido pelo usuário, as métricas de Distância de Hausdorff adaptada e coeficiente de Dice foram combinadas em um único valor real Q_S , de acordo com a equação 35.

$$Q_S = (HDa \cdot Dice_S)^2 \quad (35)$$

O valor de $HDa \cdot Dice_S$ na equação 35 é levado à segunda potência para melhor uso do intervalo $[0, 1]$, já que foi observado experimentalmente que Q_S tende a valores próximos de 1 rapidamente. Q_S é usado como métrica de *fitness* para os operadores de seleção: Neste contexto, a maior variabilidade dos valores de Q_S também proporciona melhor pressão seletiva, ajudando a convergência dos algoritmos genéticos.

6. RESULTADOS E DISCUSSÕES

6.1. Validação da métrica combinada

A Fig. 23, bem como a Tab. 7 indicam resultados de alguns testes nos quais um contorno em vermelho, e um correspondente contorno *gold-standard* em verde são comparados via coeficiente de Dice, Distância de Hausdorff direcionada (HD) e Distância de Hausdorff direcionada e adaptada *HDa*, considerando uma imagem de 512x512 pixels (ver seção 5.7.2), e métrica combinada Q_S (ver seção 5.7.3).

Tabela 7 – Comparação das métricas de avaliação de segmentação.

Teste	Dice	HD	HDa	Q_S
A	0,824068	22,3452	0,969140	0,6378210
B	0,824068	25,0000	0,965473	0,6330040
C	0,175838	67,3593	0,906972	0,0254341
D	0,175838	54,9017	0,924177	0,0264082
E	0,206646	76,1622	0,894815	0,0341917
F	0,206646	66,0298	0,908808	0,0352695
G	0,918681	4,0000	0,994476	0,8346750
H	0,918681	127,0350	0,824556	0,5738120
I	0,945203	3,2426	0,995522	0,8854240
J	0,945203	65,7308	0,909221	0,7385660

Nota-se que Q_S é primariamente influenciada pelo coeficiente de Dice, mas apresenta alguns desvios proporcionados por *HDa* que confere à métrica duas principais vantagens:

- Nos testes C e D da Fig. 23, o contorno em vermelho e verde (*gold-standard*) não apresentam sobreposição, mas recebem um valor não-nulo para coeficiente de Dice. Tal situação é penalizada mais severamente em Q_S , porém, devido ao fato de que a Distância de Hausdorff penaliza tal distância entre contornos;
- Nos testes G, H, I e J da Fig. 23, a área de intersecção entre o contorno inicial e *gold-standard* é proporcionalmente alta, conferindo alto valor de coeficiente de Dice. A distância de Hausdorff penaliza os desvios no formato dos contornos, porém.

Figura 23 – Cenários para comparação das métricas de avaliação de segmentação. Em cada um, de A a J, a cor verde indica o contorno de referência *gold-standard*, e a cor vermelha indica um outro contorno qualquer. Fonte: Daniel Coelho

Nota-se que Q_S consegue capturar de forma objetiva, pelo menos no conjunto de contornos testados, uma única medida normalizada de similaridade razoável, por par de segmentações. Nota-se porém (especialmente nos resultados dos testes G e H), que a métrica Q_S é assimétrica, por consequência do uso exclusivo da distância de Hausdorff direcionada (ver seção 5.7.2). Isso é uma desvantagem considerável, especialmente pelo fato que a situação menos penalizada do par é a situação na qual o contorno inicial apresenta um desvio em relação ao *gold-standard* (correspondendo aos testes G, I), que

potencialmente ocorre com mais frequência durante uma segmentação. Como já discutido, porém, a implementação da distância de Hausdorff completa (equação 31) acarretaria em perdas significativas em performance, e a combinação com o coeficiente de Dice ameniza o impacto de tal problema.

6.2. Validação da implementação de DRLSE em CUDA e C++

Com o objetivo de comparar as implementações do algoritmo DRLSE em CUDA e C++ com a implementação de referência em MATLAB (Li, 2010b), uma imagem DICOM foi segmentada usando as mesmas condições iniciais e parâmetros de segmentação. Esta foi exportada em um formato binário de números de ponto flutuante de 32 bits, eliminando potenciais discrepâncias devido a diferentes pré-processamentos do arquivo DICOM pelas três implementações. O conjunto nível zero, após 1000 iterações do algoritmo de segmentação, é exibido na Fig. 24.

Inspeção visual sugere uma correspondência satisfatória dos resultados obtidos das três segmentações. As pequenas discrepâncias observadas, principalmente a região observada na Fig. 24C, podem ser explicadas por dois fatores principais: O primeiro e mais significativo fator é a diferença de comportamento em cálculos que dependem de valores vizinhos, quando realizados nas extremas bordas da imagem. Como o algoritmo DRLSE usa muitas destas operações, a influência dessa discrepância propaga-se a partir das bordas da imagem ao longo das iterações. Por exemplo: O procedimento para cálculo de gradiente em MATLAB, *grad*, usa diferenças simples nos pixels das bordas (ao invés das diferenças centrais que usa normalmente). Por outro lado, o procedimento *del2* em MATLAB extrapola linearmente as diferenças segundas dos pixels interiores quando calculando os valores nas bordas. Em contraste, todas as operações em imagens na implementação em C++ lidam com os pixels das bordas truncando os índices dos pixels a valores válidos. Algo similar é feito para a implementação em CUDA, com o uso do modo *cudaBoundaryModeClamp* ao amostrar os dados. Tal abordagem foi escolhida para a implementação em CUDA por motivos de performance, já que qualquer

tipo de divergência de comportamento de pixels vizinhos em um mesmo *warp* é altamente não recomendado.

A segunda principal fonte de discrepância entre as implementações em CUDA e C++ e a implementação de referência, é o fato de que as primeiras usam exclusivamente números de ponto flutuante de 32 bits, em quanto que MATLAB usa números de ponto flutuante de 64 bits. Apesar da perda de precisão, tal escolha foi feita por três motivos:

- Efetivamente diminui pela metade o consumo de memória;
- Necessita de menos tempo para transferir e recuperar dados da GPU (em grande parte em função do motivo anterior);
- A GPU utilizada apresenta performance teórica de 4,375 trilhões de operações de números de ponto flutuante por segundo (TFLOPS) para valores de 32 bits, enquanto que apenas 0,137 TFLOPS para valores de 64 bits (uma diferença de quase 32 vezes).

Figura 24 – A: Imagem DICOM utilizada para validação. Fonte: ERICKSON, 2016; B: Região segmentada ampliada. O quadrado branco indica o contorno inicial, e o círculo lilás indica a região ampliada em C. Tracejado vermelho: Conjunto nível zero obtido da implementação em MATLAB após 1000 iterações; Linha contínua verde: Análogo para a implementação em CUDA; Linha pontilhada em azul: Análogo pra implementação em C++ serial.

6.3. Otimização da implementação de DRLSE em CUDA

De um modo geral, despachar um *kernel* para execução é uma operação com custo significativo, de modo que durante a otimização do algoritmo, busca-se a minimização do número de *kernels* necessários. A versão otimizada do algoritmo DRLSE em CUDA consiste em três execuções de *kernels* por iteração, sendo o LevelSetKernel descrito no algoritmo 4 o principal passo de iteração.

Um outro exemplo do custo de despacho de *kernels* pode ser observado no passo 11 do algoritmo 4. Ali, a função *DistRegPre*, descrita no algoritmo 5, é usada diversas vezes. Execuções do algoritmo 5 em pixels próximos levariam ao cálculo repetido de *DistRegPre* para cada pixel de *imagem_phi_grad*. O leitor pode concluir que isso apresenta uma oportunidade para otimização, e pode decidir utilizar uma imagem intermediária para armazenar os valores de *DistRegPre* para todos os pixels de *imagem_phi_grad* anterior à execução do algoritmo 5. Tal alteração não apenas aumenta o consumo total de memória do algoritmo (devido à imagem intermediária), como também acarreta em uma perda global de performance. Isso ocorre pois os custos adicionais associados ao despacho de outro *kernel*, junto com o custo de ler e escrever a tal imagem secundária, mais do que compensa os ganhos em performance relacionados à eliminação dos cálculos repetidos de *DistRegPre*.

Uma outra decisão feita em prol de performance que deve ser mencionada é a estrutura de dados escolhida. Ao longo da descrição dos algoritmos se usa o termo “imagem” como um prefixo genérico, supostamente indicando algum tipo de vetor bidimensional. No programa em CUDA em si, porém, não são utilizados simples vetores, mas sim objetos denominados *cudaSurfaceObjects*. Tais objetos são indicados para processamento de imagens em geral, por permitirem escrita e leitura, apresentarem acesso simples sem filtragem ou interpolação de hardware, e principalmente apresentam cache automático baseados em localidade 2D. Esse último significa que se uma *thread* deseja acessar um pixel de uma imagem armazenada em memória, uma curva preenchedora de espaço em 2D é usada para armazenar em cache não apenas tal pixel,

mas seus vizinhos próximos nas duas dimensões, acelerando seguintes acessos a pixels vizinhos.

Finalmente, deve-se mencionar que o número de iterações do algoritmo DRLSE não foi exposto nem ao usuário, nem ao processo de treinamento. Ao longo desta dissertação, o número de iterações foi sempre fixo no valor 1000, empiricamente determinado satisfatório, e suficiente para que a influência de um *spel* em um canto de uma imagem 512x512 se propague ao canto oposto via operações de vizinhança (como convoluções, operadores gradiente e laplaciano, entre outros).

Adicionalmente, foi observado que a influência do número de iterações é intimamente relacionada ao (e pode ser replicada pelo) efeito dos demais parâmetros. Em outras palavras, um conjunto de parâmetros treinado com 1000 iterações do algoritmo DRLSE pode não apresentar resultados satisfatórios com mais ou menos iterações. Por exemplo: Um conjunto de parâmetros treinado em 1000 iterações para um contexto de segmentação em que o contorno inicial se encontra próximo do *gold-standard* pode apresentar bons resultados com um certo valor de α (tendência à expansão ou contração). Em 500 iterações, porém, o mesmo valor de α potencialmente não permitirá tempo suficiente para que o contorno se expanda à posição alvo, levando a um resultado pior. Em 2000 iterações, por outro lado, o efeito do mesmo valor de α pode fazer com que o contorno ultrapasse sua posição ideal.

Pode-se argumentar, porém, que a disponibilidade dos parâmetros τ (timestep), bem como o modo de execução tardio em geral (ver seção 5.5), providenciam ao usuário e processo de treinamento controle sobre os mesmos graus de liberdade que seriam providenciados pelo controle do número de iterações. Adicionalmente, estes não proporcionam potenciais prejuízos de implementação e uso relacionados a um tempo imprevisível e variável do processo de aprendizado.

6.3.1. *Occupancy*

Baseado em análises de performance com NVVP, foi determinado experimentalmente que o número ideal de *threads* por bloco para o LevelSetKernel

(LSK), na NVIDIA GTX 1060 6GB, é de 64, em contraste com o valor máximo para tal hardware de 1024. Em outros contextos, com *kernels* mais simples, a GPU conseguiria executar mais blocos em um mesmo SM para compensar, ainda assim atingindo 100% de *occupancy*. Para o LSK, porém, de acordo com as informações providenciadas pelo NVVP, têm-se que a *occupancy* máxima para o LSK é de teoricamente 62,5%, com valores observados de 59%.

Tal discrepância em comparação ao valor alvo de 100% é explicado pelo fato de que o LSK usa 48 registros por *thread*. Com 64 *threads* por bloco, o *kernel* totaliza então 3072 registros por bloco. Dado o limite do hardware de 65536 *threads* por SM, temos que o driver consegue agendar no máximo 20 blocos simultâneos por SM. Esses 20 blocos, cada um com 64 *threads*, totalizam 1280 *threads* simultâneas pra cada SM. A razão entre tal número e o número máximo de *threads* por SM permitido para esse hardware (2048) é o que leva à máxima teórica *occupancy* de 62,5%.

A principal recomendação em situações como essa, em que o uso de registros limita *occupancy* de 100%, é de limitar o número de registros que o compilador pode utilizar para cada *thread*. A ideia é que com menos registros por *thread*, o driver conseguiria despachar mais blocos para cada SM, permitindo uma maior *occupancy*. Por outro lado, limitar o número de registros faz com que o compilador seja forçado a utilizar o cache L1 para armazenar parte dos dados usados ao longo do *kernel*, acarretando em uma perda de performance. Depende do contexto, então, se o aumento de *occupancy* vale o custo da restrição do número de registros por *thread*.

Para investigar o efeito da restrição do número de registros por *thread*, foram realizadas diversas segmentações de uma imagem 512x512, com restrições desde 48 (caso ideal) à 32 registros por *thread*. Os três *kernels* usados pela implementação do algoritmo DRLSE (LSK, NormalizedGradientKernel e LaplaceKernel) foram cronometrados, bem como o tempo total para execução de 1000 iterações completas. Estes resultados são exibidos na Tab. 8.

Na Tab. 8, a coluna “Reg. Max.” indica a restrição de registros configurada. 48 corresponde ao número de registros usados naturalmente pelo LSK, o maior valor dentre os 3 *kernels* observados. O compilador pode não necessariamente usar a totalidade

desse valor, porém, podendo decidir usar menos registros do que o disponível. O número de registros de fato usados pelo LSK é indicado na coluna “Reg. Usados LSK”. As três colunas “Tempo LSK”, “Tempo NGK” e “Tempo LK” correspondem ao tempo médio usados pelos três *kernels* (respectivamente LSK, NormalizedGradientKernel e LaplaceKernel) após 1585000 execuções de cada. Os valores são providenciados pelo software NVVP, de modo que o desvio padrão não é disponível.

Tabela 8 – *Occupancy* e performance como uma função da restrição de registros.

Reg. Max.	Reg. Usados LSK	Occ. teórica (%)	Occ. observada (%)	Tempo LSK (μ s)	Tempo NGK (μ s)	Tempo LK (μ s)	1000 iterações (ms)
48	48	62,5	59,2	89,081	37,029	25,772	180,27
46	46	62,5	59,2	96,633	40,095	27,240	179,71
44	40	75,0	70,3	102,96	38,621	23,661	181,52
42	40	75,0	70,5	104,75	38,026	26,425	183,60
40	40	75,0	70,3	111,40	39,729	24,736	189,69
38	38	75,0	70,7	110,43	40,055	26,626	191,60
36	32	100,0	90,4	130,84	39,842	26,606	206,55
34	32	100,0	90,9	128,31	40,649	26,801	208,94
32	32	100,0	91,5	129,60	40,557	26,607	204,58

Note que tanto para estes testes, quanto para todos os demais testes descritos nesta dissertação nos quais performance é relevante, o tempo usado para alocar, transferir e retornar memória da GPU foi ignorado. Adicionalmente, os dados foram sempre coletados após um período significativo de “aquecimento” da GPU. Isso é usual no campo (CHENG, 2014), e deve-se ao fato de que muitos destes *kernels* são compilados em regime *just-in-time*, de modo que a primeira execução destes frequentemente consome dezenas de vezes mais tempo do que o consumido em um caso comum.

Os dados descritos na Tab 8. mostram que conforme o número de registros se torna mais restrito, *occupancy* teórica e observada (“Occ. teórica” e “Occ. observada”) de fato aumentam como esperado, mas a performance em geral tende a piorar, o que indica que este não é um bom método de otimizar o programa desenvolvido. Note também que a performance do NormalizedGradientKernel e LaplaceKernel não é tão afetada pela restrição de registros quanto a performance do LSK. Isso se dá pelo fato de que os dois

primeiros usam 22 e 20 registros por *thread*, respectivamente. É possível notar também que o compilador não acompanha exatamente a restrição do número de registros, mas decide alocar grupos de até 6 registros de uma única vez. Seria necessário analisar detalhadamente o código *assembly* gerado pelo compilador para entender o motivo por trás de tais decisões, mas tal fato não afeta as conclusões que podem ser tomadas de tais dados: De modo não-intuitivo, *occupancy* de 62,5% é o ótimo em performance para esta implementação em CUDA do algoritmo DRLSE para o hardware utilizado.

6.3.2. Efeito da escolha de GPU

Um dos objetivos de uma biblioteca como CUDA é de conferir uma camada de abstração entre o programa e o hardware, de modo que o programa desenvolvido em CUDA funciona de modo análogo em qualquer placa que suporte uma *Compute Capability* superior ou igual à 6.1, e cumpra com os requisitos de memória (ver seção 6.10). Apesar disso, como indicado na seção anterior, a performance final do programa está intimamente relacionada a algumas configurações (como número ideal de *threads* por bloco, ou número de registros por *thread*), de modo que é difícil prever o tempo de execução do programa em outra placa de vídeo. Apesar disso, a placa utilizada (NVIDIA GTX 1060 6GB) não é um modelo exótico ou particular, sendo lançada em julho de 2016 com público alvo o consumidor comum, de modo que placas de vídeo disponíveis hoje potencialmente alcançariam performance igual ou superior às reportadas nessa dissertação para o programa desenvolvido.

6.4. Escolha de metaparâmetros para o algoritmo genético

A eficiência de cada operador genético, e do algoritmo genético como um todo, depende altamente da aplicação. Apesar de ser possível fazer algumas afirmações genéricas, no campo de algoritmos genéticos o teste empírico é essencial. Essa realidade é ainda mais relevante no caso desta dissertação, no qual o algoritmo genético desenvolvido deve ser usado para contextos de segmentação variados: Segmentos com

bordas distintas e bordas graduais; Segmentos convexos e com formato complexo; Segmentos únicos ou fragmentados; entre outros.

Apesar disso, esse trabalho parte da hipótese de que a variabilidade dos contextos de segmentação não necessariamente impede a existência de um conjunto de operadores genéticos que apresente performance satisfatória na maioria dos casos.

Com o objetivo de encontrar tal conjunto de operadores genéticos, foram realizados diversos testes, avaliando o desempenho do processo de treinamento. Infelizmente, a grande variedade de operadores genéticos (cada um por sua vez com seus próprios parâmetros) impede uma análise exaustiva de todas as combinações em uma escala de tempo razoável. Ao invés disso, uma combinação inicial satisfatória de operadores foi configurada (indicada na Tab. 9).

Tabela 9 – Operadores genéticos usados como padrão.

Tipo de operador	Operador escolhido
Seleção	Ranking Exponencial (base 0,5)
Recombinação	Uniforme (probabilidade $P_C = 75\%$)
Mutação	Uniforme (probabilidade $P_M = 20\%$)
Elitismo	2 indivíduos

A escolha final dos operadores genéticos objetiva um AG que seja capaz de rapidamente convergir em um bom valor final de *fitness* (no caso correspondendo a métrica Q_S), sem fazê-lo de modo susceptível ao fenômeno de convergência prematura.

Para alcançar tal objetivo, a estratégia por traz da escolha de operadores inicial na Tab. 9 é a de garantir robustez e exploração com o uso do elitismo (no caso garantindo que dois dos melhores indivíduos sejam movidos diretamente à geração seguinte), simultaneamente estimulando a exploração com operadores de mutação e recombinação disruptivos. Adicionalmente, o operador de Ranking Exponencial foi escolhido pois apresenta algumas propriedades teoricamente superiores aos operadores de ranking linear, torneio e truncamento, como a menor influência na variância da distribuição de *fitness* dos indivíduos, o que ajuda a prevenir a convergência prematura (BLICKLE, 1995).

Com o intuito de determinar se algum operador é particularmente efetivo nas situações de segmentação estudadas, cada operador inicial foi substituído por todas as suas alternativas (indicadas na Tab. 6, seção 5.6) de modo sistemático. Em casos em que o operador possui algum tipo de parâmetro (como a taxa de mutação P_M ou de recombinação P_C), diversas versões de cada operador foram examinadas quando relevante. Cada grupo de operadores foi então testado em 3 diferentes contextos de segmentação, 10 vezes cada. Os resultados médios e os desvios-padrões são exibidos nas Fig. 25 a 34. Para facilitar a visualização dos dados, os operadores de recombinação foram divididos em três grupos, e plotados separadamente.

Em todos os casos, o número de indivíduos foi configurado como 30. Como usual no campo de algoritmos genéticos, o número ideal de indivíduos depende dos operadores e aplicação, existindo diversas recomendações a favor de altos e baixos valores (HAUPT, 2000; DE JONG, 1975; GOLDBERG, 1989; GREFENSTETTE, 1986). O tamanho 30 confere uma população de tamanho entre baixo e médio, e tenta alcançar um equilíbrio dentre todas as recomendações, de um modo que faça sentido para o tamanho dos cromossomos utilizados. A relativamente pequena população faz com que o AG seja especialmente suscetível à convergência prematura, ressaltando a importância de operadores de mutação e recombinação disruptivos.

Após alguns testes com 30 indivíduos, notou-se que 60 gerações é tempo o suficiente para que a maioria dos algoritmos genéticos convirjam em algum resultado estacionário. Tais valores de 30 indivíduos e 60 gerações foram usados em todos os treinamentos realizados nessa dissertação.

Finalmente, note que o operador de mutação gaussiano foi descartado de tais testes, pois um σ_m pequeno não seria satisfatoriamente disruptivo, enquanto que um σ_m grande apresentaria efeito comparável à mutação uniforme. Adicionalmente, o operador de seleção por truncamento é descartado pois apresenta elevada perda de diversidade quando comparado aos demais (BLICKLE, 1995).

Figura 25 – Análise de operadores de seleção para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, Q_s médio é exibido, e as 20 últimas gerações são ampliadas à direita.

Figura 26 – Análise de operadores de seleção para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, o desvio padrão de Q_s é exibido, e as 20 últimas gerações são ampliadas à direita.

Figura 27 – Análise de operadores de recombinação (grupo 1) com diferentes valores de P_C para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, Q_s médio é exibido, e as 20 últimas gerações são ampliadas à direita.

Figura 28 – Análise de operadores de recombinação (grupo 1) com diferentes valores de P_c para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, o desvio padrão de Q_s é exibido, e as 20 últimas gerações são ampliadas à direita. Operadores de recombinação dos demais grupos são exibidos em cinza.

Figura 29 – Análise de operadores de recombinação (grupo 2) com diferentes valores de P_c para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, Q_s médio é exibido, e as 20 últimas gerações são ampliadas à direita. Operadores de recombinação dos demais grupos são exibidos em cinza.

Figura 30 – Análise de operadores de recombinação (grupo 2) com diferentes valores de P_c para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, o desvio padrão de Q_s é exibido, e as 20 últimas gerações são ampliadas à direita. Operadores de recombinação dos demais grupos são exibidos em cinza.

Figura 31 – Análise de operadores de recombinação (grupo 3) com diferentes valores de P_c para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, Q_s médio é exibido, e as 20 últimas gerações são ampliadas à direita. Operadores de recombinação dos demais grupos são exibidos em cinza.

Figura 32 – Análise de operadores de recombinação (grupo 3) com diferentes valores de P_c para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, o desvio padrão de Q_s é exibido, e as 20 últimas gerações são ampliadas à direita. Operadores de recombinação dos demais grupos são exibidos em cinza.

Figura 33 – Análise de operadores de mutação com diferentes valores de P_M para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, Q_s médio é exibido, e as 20 últimas gerações são ampliadas à direita.

Figura 34 – Análise de operadores de mutação com diferentes valores de P_M para os contextos de segmentação ‘cérebro1’, ‘cérebro2’ e ‘células1’. Em cada linha, o desvio padrão de Q_S é exibido, e as 20 últimas gerações são ampliadas à direita.

Observando os resultados das Fig. 25 a 34, nota-se que todos os operadores levaram a resultados relativamente satisfatórios. Observa-se também certa quantidade de ruído no formato das curvas, devido ao impacto da geração aleatória de indivíduos. De um modo geral, porém, é possível notar algumas tendências:

- Os operadores de seleção baseado em torneio (especialmente com 4 indivíduos por torneio) apresentam bons valores finais de *fitness* e convergência rápida, paralelamente ao baixo desvio-padrão;
- Alguns operadores apresentam resultados superiores em alguns contextos de segmentação, e inferiores em outros (como recombinação uniforme com $P_C = 1,0$). Uma potencial justificativa para tal fenômeno seria discrepâncias na topologia do espaço de parâmetros de cada contexto de segmentação, requerendo mais ou menos exploração para ser garantir sucesso (ver seção 6.5), assim favorecendo operadores com uma razão exploração/exploração específicos. Independentemente, operadores marcadamente sensíveis ao contexto de segmentação não são bons candidatos à configuração final do sistema, que requer maior robustez;
- O operador de mutação uniforme com $P_M = 0,2$ acarreta em resultados finais significativamente superiores, e apresenta convergência um pouco mais rápida que as alternativas.

A configuração final do sistema então trocará o operador de seleção pelo baseado em torneio, com 4 indivíduos por torneio. O operador de mutação com $P_M = 0,2$ da configuração inicial é significativamente superior às alternativas e continuará sendo usado, enquanto que as alternativas para o operador de recombinação são todas comparáveis ao operador usado inicialmente, sendo melhores em alguns contextos e piores em outros, não sugerindo alteração. A configuração final do sistema é resumida na Tab. 10.

Tabela 10 – Metaparâmetros finais para o algoritmo genético.

Metaparâmetro	Valor
Operador de seleção	Torneio (4 indivíduos)
Operador de recombinação	Uniforme ($P_C = 75\%$)
Operador de mutação	Uniforme ($P_M = 20\%$)
Elitismo	2 indivíduos
Número de gerações	60
População	30 indivíduos

6.5. Varredura do espaço de parâmetros

Com o intuito de se obter uma visão geral da complexidade do espaço de parâmetros do algoritmo DRLSE, o programa desenvolvido foi utilizado para realizar uma varredura exaustiva do mesmo, para três diferentes contextos de segmentação. Nos três casos, temos que todos os parâmetros foram fixados, exceto λ e α , que variaram no intervalo $(0, 30]$ e $[-30, 30]$, em incrementos de 0,2. De modo idêntico ao processo de treinamento, as segmentações resultantes após 1000 iterações foram comparadas com os correspondentes *gold-standard* de acordo com a métrica combinada (ver seção 5.7.3). O valor de *fitness* resultante para cada combinação de λ e α é plotado nas Fig. 35 a 39.

Os valores dos parâmetros fixados correspondem ao conjunto de parâmetros de melhor *fitness* dentre 10 consecutivos processos de treinamento para cada contexto de segmentação. Tais valores são indicados na Tab. 11. Note, porém, que apesar de indicados, os valores de λ e α são obviamente ignorados nas Fig. 35 a 39, indicando os valores de *fitness* em função destes.

Tabela 11 – Conjunto de parâmetros com melhor *fitness* por contexto de segmentação.

Parâmetro	Contexto de segmentação				
	abdome	células1	cérebro1	cérebro2	pelvis1
<i>fitness</i>	0,888400	0,832085	0,911127	0,856999	0,958805
μ	0,200000	0,200000	0,200000	0,200000	0,200000
λ	18,979300	3,073750	0,978996	9,737610	11,993300
α	6,258330	-28,404900	2,709940	13,867600	-27,150600
σ	2,641320	3,993710	0,084269	2,478540	0,225262
τ	1,250000	1,160650	1,250000	1,250000	0,680637
i_l	0,930242	0,753687	0,911646	0,614338	0,897564
μ_l	0,200000	0,200000	0,200000	0,200000	0,200000
λ_l	4,976470	29,761100	29,622800	27,647800	26,153600
α_l	-29,091500	-25,553300	10,816300	18,360000	-23,970200
τ_l	0,794773	1,250000	0,480837	1,250000	1,250000

Figura 35 – Varredura do espaço de parâmetros para o contexto de segmentação ‘abdome’. Cada pixel descreve o valor de *fitness* obtido da segmentação usando os correspondentes valores de α , λ , e os demais valores indicados na Tab. 11. O ponto em vermelho indica os valores de α e λ mostrados na Tab. 11.

Figura 36 – Varredura do espaço de parâmetros para o contexto de segmentação ‘celulas1’. Cada pixel descreve o valor de *fitness* obtido da segmentação usando os correspondentes valores de α , λ , e os demais valores indicados na Tab. 11. O ponto em vermelho indica os valores de α e λ mostrados na Tab. 11.

Figura 37 – Varredura do espaço de parâmetros para o contexto de segmentação ‘cerebro1’. Cada pixel descreve o valor de *fitness* obtido da segmentação usando os correspondentes valores de α , λ , e os demais valores indicados na Tab. 11. O ponto em vermelho indica os valores de α e λ mostrados na Tab. 11.

Figura 38 – Varredura do espaço de parâmetros para o contexto de segmentação ‘cerebro2’. Cada pixel descreve o valor de *fitness* obtido da segmentação usando os correspondentes valores de α , λ , e os demais valores indicados na Tab. 11. O ponto em vermelho indica os valores de α e λ mostrados na Tab. 11.

Figura 39 – Varredura do espaço de parâmetros para o contexto de segmentação ‘pelvis’. Cada pixel descreve o valor de *fitness* obtido da segmentação usando os correspondentes valores de α , λ , e os demais valores indicados na Tab. 11. O ponto em vermelho indica os valores de α e λ mostrados na Tab. 11.

A diferença entre a topologia dos espaços de parâmetros indica como que a escolha de tais parâmetros é um processo altamente dependente da situação de segmentação. Em um contexto de segmentação como ‘pelvis1’, é mais fácil encontrar uma combinação de parâmetros que leve a uma boa segmentação do que em um contexto como ‘cerebro2’, por exemplo. Vale ressaltar que as Fig. 35 a 39 indicam planos do espaço de parâmetros que contêm o ponto de maior *fitness* encontrado, apresentando então grande viés para bons valores de *fitness*. No caso geral, a grande maioria do espaço de parâmetros para tais contextos de segmentação acarreta em *fitness* pior do que o retratado.

As Fig. 35 e 38 especialmente mostram as dificuldades de se encontrar o conjunto de parâmetros ideal para o algoritmo DRLSE em algumas situações: Além de boa parte do espaço de parâmetros levar a uma segmentação com *fitness* próximo de zero, mesmo em regiões onde o *fitness* é em média razoável, um certo “ruído” pode existir, que faz com que combinações de parâmetros similares levem a resultados altamente distintos. Tais imagens ressaltam a importância de sistemas como o desenvolvido neste trabalho.

6.6. Exemplos de segmentações usando conjuntos de parâmetros gerados

Essa seção dos resultados exhibe nas Fig. 40, 41 e 42 as segmentações resultantes ao utilizar o algoritmo DRLSE com o melhor conjunto de parâmetros obtido para cada contexto de segmentação, após 10 processos de treinamento. Os valores dos parâmetros, bem como o valor de *fitness* correspondente, são indicados na Tab. 11.

Figura 40 – Resultado da segmentação com parâmetros obtidos pelo processo de treinamento. Vermelho: Contorno inicial; Azul: Contorno final após segmentação com algoritmo DRLSE; Verde: Contorno *gold-standard*. Acima: Contexto ‘abdome’ e respectiva vista ampliada. A baixo: Contexto ‘células1’ e respectiva vista ampliada.

Figura 41 – Resultado da segmentação com parâmetros obtidos pelo processo de treinamento. Vermelho: Contorno inicial; Azul: Contorno final após segmentação com algoritmo DRLSE; Verde: Contorno *gold-standard*. Acima: Contexto ‘cérebro1’ e respectiva vista ampliada. Abaixo: Contexto ‘cérebro2’ e respectiva vista ampliada.

Figura 42 – Resultado da segmentação com parâmetros obtidos pelo processo de treinamento para o contexto de segmentação ‘pelvis1’. Vermelho: Contorno inicial; Azul: Contorno final após segmentação com algoritmo DRLSE; Verde: Contorno *gold-standard*. A direita: Vista ampliada.

Apesar da alta capacidade de segmentação do algoritmo DRLSE, e da relativamente boa qualidade dos contornos segmentados, é importante lembrar que o processo de segmentação (e indiretamente o processo de treinamento) possui a limitação de que opera sobre a imagem indicadora de bordas $g(x, y)$. O movimento e localização final do conjunto nível zero, então, estão restritos à informação providenciada por esta. No caso dessa dissertação, a imagem indicadora de bordas trata-se de um filtro de detecção de bordas, e todos os contextos de segmentação estudados envolveram a segmentação de bordas visíveis na imagem. O sistema desenvolvido não é capaz, então, de produzir conjuntos de parâmetros capazes de fazer com que o conjunto nível zero convirja a contornos *gold-standard* de formatos arbitrários, simplesmente porque tais conjuntos de parâmetros não necessariamente existem. Tal fato pode ser especialmente visualizado na restante diferença entre os contornos finais e os *gold-standard* nos contextos de segmentação exibidos.

Adicionalmente, outro fato que deve ser ressaltado é que os contornos *gold-standard* foram traçados pelo autor dessa dissertação, e não necessariamente seguem valores de intensidade consistentes ou delineiam estruturas anatômicas distintas, potencialmente dificultando o processo de treinamento.

6.7. Análise do gold-standard

Com o uso de *phantoms*, essa seção visa analisar a métrica escolhida, bem como os resultados do processo de treinamento, quando pode-se comprovar que o *gold-standard* é correto. Tal teste é relevante no contexto deste projeto pois os demais *gold-standards* foram traçados em imagens anatômicas pelo autor da dissertação, que não possui treinamento médico.

O simples *phantom* utilizado, bem como os contornos *gold-standard* e inicial são apresentados na Fig. 43. Foram realizados 10 treinamentos para tal contexto de segmentação, e a média e desvio padrão da métrica combinada Q_S resultante são apresentados na Tab. 12. O mesmo foi feito para contextos de segmentação similares, adicionando-se ruído normal com valores crescentes de desvio padrão. Um exemplo de tais contextos é mostrado na Fig. 43B, e os resultados de todos os testes com ruído são também apresentados na Tab. 12.

Figura 43 – Contextos de segmentação com *phantom*. A: Sem ruído adicional. Em vermelho: Contorno inicial; Em verde: Contorno *gold-standard*. B: *phantom* adicionado de ruído normal com desvio padrão de valor 250. Fonte: Daniel Coelho

Note que, como nos demais contextos de segmentação estudados neste projeto, o *phantom* sem ruído apresenta *spels* com valores de intensidade mapeadas ao intervalo $[0, 1000]$. Mais especificamente, a região clara na Fig. 43A apresenta pixels com intensidade 1000, e a região escura apresenta pixels com intensidade 0. O ruído foi sempre aplicado após tal mapeamento. Por outro lado, note que a Fig. 43B apresenta escala de cinza renormalizada, para visualização sem truncamento.

Tabela 12 – Médias e desvios-padrões de Q_S para 10 treinamentos com *phantom*.

Desvio padrão do ruído aplicado	Q_S
0	$0,9822848 \pm 0,0020080$
50	$0,9648097 \pm 0,0045020$
100	$0,9478817 \pm 0,0071887$
250	$0,9193769 \pm 0,0036660$
500	$0,8164235 \pm 0,0173212$
1000	$0,5767016 \pm 0,0131837$
2000	$0,3127273 \pm 0,0013601$

Observa-se que a métrica combinada apresenta valor próximo do máximo (1,0) para o contexto de segmentação do *phantom* sem ruído, com resultados piores conforme a intensidade do ruído aumenta. A Fig. 44 apresenta o contorno final obtido no contexto de segmentação sem ruído, junto com uma versão ampliada. O fato da métrica Q_S não apresentar valor máximo 1,0 no contexto de segmentação sem ruído pode ser justificado ao observar na Fig. 44B como o conjunto nível zero é incapaz de penetrar pequenas concavidades do *phantom*.

Figura 44 – A: Resultado do treinamento com *phantom* sem ruído. Em vermelho: Contorno inicial; Em azul: Contorno final. B: Vista ampliada do canto superior-direito de A. Fonte: Daniel Coelho

6.8. Análise de consistência

Com o intuito de avaliar a consistência do conjunto de parâmetros produzido, foram realizados 10 treinamentos para cada contexto de segmentação estudado, e os conjuntos de parâmetros resultantes foram compilados. A Tab. 13 indica a média de cada parâmetro por contexto, junto com os respectivos desvios-padrões. Note que μ e μ_l são fixos em 0,2 durante o processo de treinamento.

Tabela 13 – Médias e desvios-padrões de 10 treinamentos para cada contexto.

Parâmetro	Contexto de segmentação				
	abdome	células1	cérebro1	cérebro2	pelvis1
<i>fitness</i>	0,851865 ± 0,017452	0,800255 ± 0,017952	0,796086 ± 0,035829	0,891279 ± 0,016049	0,934416 ± 0,019178
μ	0,200000 ± 0,000000	0,200000 ± 0,000000	0,200000 ± 0,000000	0,200000 ± 0,000000	0,200000 ± 0,000000
λ	18,320548 ± 8,762879	17,827755 ± 6,262234	13,335831 ± 7,218077	4,425005 ± 5,587597	9,835608 ± 8,508700
α	7,302109 ± 3,974829	-26,130320 ± 2,312908	14,321960 ± 2,335275	3,975069 ± 2,759029	-12,097015 ± 10,195676
σ	2,471242 ± 1,152301	5,862435 ± 2,390224	3,569741 ± 2,037095	1,255508 ± 0,834370	0,641426 ± 0,673139
τ	0,974479 ± 0,250679	1,241065 ± 0,026805	1,250000 ± 0,000000	1,061765 ± 0,332054	1,083917 ± 0,229151
i_l	0,937736 ± 0,121119	0,965279 ± 0,076678	0,829557 ± 0,281470	0,982329 ± 0,035342	0,826478 ± 0,223873
μ_l	0,200000 ± 0,000000	0,200000 ± 0,000000	0,200000 ± 0,000000	0,200000 ± 0,000000	0,200000 ± 0,000000
λ_l	15,867813 ± 9,666626	16,418797 ± 10,907960	20,761378 ± 6,354978	19,325463 ± 9,795913	18,222342 ± 10,195208
α_l	2,505689 ± 22,055938	0,317463 ± 18,470864	5,546527 ± 13,523883	-2,918424 ± 12,772287	-7,246225 ± 16,441957
τ_l	1,204477 ± 0,136568	1,202319 ± 0,143042	1,232321 ± 0,053037	1,107136 ± 0,286779	1,047556 ± 0,337108

É relevante exibir e discutir tais dados pois estes explicitam um comportamento não intuitivo dos conjuntos de parâmetros gerados: no primeiro momento estes sugerem que os conjuntos de parâmetros produzidos pelo AG são relativamente inconsistentes, e que o sistema é incapaz de encontrar o máximo global do espaço de parâmetros. O desvio-padrão do valor de *fitness* é relativamente baixo, porém (sugerindo boa consistência), e

a elevada variabilidade nos conjuntos de parâmetros gerados é esperada, e se dá por dois motivos:

O primeiro e principal motivo é o fato de que conjuntos de parâmetros distintos (ou seja, com desvio padrão significativo) podem acarretar em um contorno final similar, com *fitness* similar. Por exemplo: Considere um contexto de segmentação que consista em um contorno inicial circular externo que deva contrair-se para alinhar a uma borda circular interna. O algoritmo DRLSE pode alcançar tal objetivo com $\alpha \ll 0$ (tendência a redução de área), com $\lambda \gg 0$ (tendência a redução de comprimento), ou combinando $\alpha < 0$ e $\lambda \gg 0$, ou ainda combinando $i_l < 1,0$ (modo tardio, descrito na seção 5.5) com $\alpha_l \ll 0$, entre outros.

O segundo motivo, relacionado ao primeiro, é o grau de liberdade conferido pelo mecanismo de parâmetros iniciais e tardios (ver seção 5.5). Quando $i_l \neq 0$ e $i_l \neq 1$, temos que o AG passa partes significativas de sua evolução em modo tardio e em modo inicial, podendo “dividir” a contribuição de cada modo arbitrariamente. Por exemplo: Considere o mesmo contexto descrito no parágrafo anterior, que necessita de contração do contorno inicial para bom valor de *fitness*. Caso $i_l = 0,5$, temos que os conjuntos de parâmetros $(\alpha = -5, \alpha_l = -15)$, $(\alpha = -15, \alpha_l = -5)$, $(\alpha = 0, \alpha_l = -20)$, $(\alpha = -20, \alpha_l = 0)$, possam todos levar ao mesmo contorno final (e com isso mesmo valor de *fitness*), apesar de significativamente inconsistentes. Adicionalmente, caso $i_l = 1$, temos que todas as versões “tardias” dos parâmetros são ignoradas (podendo então apresentar alto desvio-padrão), enquanto que o oposto ocorre caso $i_l = 0$. Caso i_l apresente significativo desvio-padrão (observado em diversos dos contextos estudados), temos que o desvio padrão dos demais parâmetros pode então ser adicionalmente “poluído”.

6.9. Análise de sensibilidade

Esta seção envolve observar o impacto do posicionamento e formato de um contorno inicial ‘A’ durante a aplicação do algoritmo DRLSE, usando um conjunto de parâmetros treinado em um contexto de segmentação com contorno inicial ‘B’. O ideal seria que o

sistema apresentasse certa robustez, de modo que mesmo caso contornos 'A' e 'B' fossem significativamente distintos, a segmentação final fosse comparável. Para isso, os contextos de segmentação 'pélvis1' e 'células1' foram segmentados com diferentes contornos iniciais, e os resultados são exibidos nas Fig. 45 e 46. Foram usadas 1000 iterações do algoritmo DRLSE, com os parâmetros indicados na Tab. 11.

Pode-se concluir a partir das Fig. 45 e 46 que o sistema apresenta certa robustez quanto ao posicionamento do contorno inicial. Devido às limitações e sensibilidade do algoritmo DRLSE, porém, tal robustez é limitada. Para obtenção de resultados similares, é necessário que o contexto de segmentação apresente distribuição de intensidade dos pixels similar ao contexto de treinamento, e que o contorno inicial se encontre à uma distância do contorno alvo similar ao observado no contexto de treinamento.

Na Fig. 45C, por exemplo, apesar de distinto e relativamente mais próximo, o contorno inicial envolve uma região similar ao contexto de treinamento. Na Fig. 45B, o contorno inicial é mais distante do objeto alvo, e passa a envolver uma borda adicional na imagem (indicada em vermelho), de modo que o contorno final obtido é diferente do observado durante o treinamento.

Adicionalmente, a Fig. 46C explicita ainda mais o efeito da variação na distância do contorno inicial ao objeto alvo: Apesar da região envolvida pelo contorno inicial ser similar ao observado no contexto de treinamento, a posição do contorno inicial é significativamente mais próxima ao objeto. A "tendência de contrair" conferido à evolução do contorno pelos parâmetros gerados durante o treinamento faz aqui com que o contorno inicial penetre no objeto alvo, gerando um contorno final incorreto.

Tais limitações do algoritmo DRLSE levantadas poderiam ser parcialmente mitigadas com a modificação do sistema implementado para realização do treinamento simultâneo em múltiplos contextos de segmentação distintos, potencialmente conferindo maior robustez ao sistema, mas acarretando em maior custo em performance.

Figura 45 – Teste de sensibilidade ao contorno inicial no contexto de segmentação ‘pelvis1’. Vermelho: Contorno inicial; Azul: Contorno final após segmentação com algoritmo DRLSE; Verde: Contorno *gold-standard*. A: Situação de treinamento; B e C: Contornos iniciais alternativos e respectivos contornos finais. A seta vermelha indica uma borda adicional envolvida pelo contorno inicial no caso B.

Figura 46 – Teste de sensibilidade ao contorno inicial no contexto de segmentação ‘células1’. Vermelho: Contorno inicial; Azul: Contorno final após segmentação com algoritmo DRLSE; Verde: Contorno *gold-standard*. A: Situação de treinamento; B e C: Contornos iniciais alternativos e respectivos contornos finais.

6.10. Dependência no contexto de segmentação

Esta seção envolve analisar o uso de conjuntos de parâmetros treinados em um contexto de segmentação 'A' em um contexto similar, porém distinto, 'B'. Difere da seção anterior ao utilizar diferentes imagens da mesma série DICOM, ou imagens de séries DICOM diferentes (e pacientes diferentes) que retratam objetos similares. Adicionalmente, os contornos iniciais foram traçados nesses novos contextos de segmentação com o intuito de que se assemelhem aos análogos no contexto de treinamento, objetivando apenas uma análise do efeito da alteração da imagem segmentada em si. Em todos os casos, foram realizadas 1000 iterações do algoritmo DRLSE com os parâmetros indicados na Tab. 11, e os resultados são apresentados nas Fig. 47 a 49.

Nota-se que é possível utilizar os valores dos parâmetros para o algoritmo DRLSE treinados em um contexto de segmentação em diferentes contextos similares, seja em imagens vizinhas da mesma série ou apenas imagem que retratem as mesmas estruturas anatômicas.

Como discutido na seção 6.6, tal comportamento possui certas limitações: O algoritmo DRLSE é sensível à distribuição de intensidades no objeto a segmentar, suas bordas e vizinhança. Pode-se observar na Fig. 48, por exemplo, que o contorno final extravasou a região alvo, comportamento este não observado no contexto de treinamento 'cérebro1', apesar do contorno inicial ser comparável. Adicionalmente, nota-se nos quadros inferiores da Fig. 49 que o contorno final penetrou a borda óssea, apesar do posicionamento comparável do contorno inicial. Ambas discrepâncias levantadas potencialmente são consequências de pequenas diferenças nas intensidades dos pixels da imagem, como uma borda óssea cujos pixels possuem intensidade relativamente menor.

Por outro lado, é possível observar nos quadros superiores Fig. 49 que mesmo se a estrutura anatômica for significativamente diferente, caso as distribuições de intensidades dos pixels no objeto a segmentar, suas bordas e vizinhança forem similares ao observado no contexto de treinamento, o algoritmo gera resultados satisfatórios. Essa

observação sugere que seria possível expandir o sistema desenvolvido para segmentar uma série de imagens com um único conjunto de parâmetros e contorno inicial bidimensional, desde que seja desenvolvido alguma metodologia para geração programática de contornos iniciais nas demais imagens da série. Essa capacidade tornaria possível, por exemplo, segmentar estruturas tridimensionais em séries de imagens, usando a mesma implementação bidimensional do algoritmo DRLSE, junto da facilidade de delimitação manual de um contorno inicial bidimensional.

Figura 47 – Testes usando o conjunto de parâmetros treinados no contexto de segmentação ‘celulas1’, indicado na Fig. 13. Vermelho: Contorno inicial; Azul: Contorno final. Acima: Imagem 415 e vista ampliada, estudo MiMM_SBiLab. Abaixo: Imagem 506 e vista ampliada, estudo MiMM_SBiLab. Fonte: GUPTA, 2019.

Figura 48 – Testes usando o conjunto de parâmetros treinados no contexto de segmentação ‘cerebro1’, indicado na Fig. 14. Vermelho: Contorno inicial; Azul: Contorno final. Imagem 165 e vista ampliada, série 801-sT1W3DTFEAX POST-99314, estudo 04-23-2000-MR BRAIN WOW CONTRAST-18837, paciente C3L-00629. Fonte: CPTAC, 2018a.

Figura 49 – Testes usando o conjunto de parâmetros treinados no contexto de segmentação ‘pelvis1’, indicado na Fig. 16. Vermelho: Contorno inicial; Azul: Contorno final. Acima: Imagem 269 e vista ampliada, série abdominallymphnodes-03108, paciente ABD_LYMPH_005; Centro: Imagem 183 e vista ampliada, série abdominallymphnodes-51545, paciente ABD_LYMPH_006. Abaixo: Imagem 206 e vista ampliada, série abdominallymphnodes-03108, paciente ABD_LYMPH_005. Fonte: ROTH, 2015.

6.11. Uso total de memória em modos de treinamento e aplicação

Para a execução de uma segmentação com a implementação do algoritmo DRLSE desenvolvida em CUDA, temos que o consumo de memória total, em bytes, é de um modo geral descrito pela equação 36.

$$Total_{DRLSE} \approx 11 * \max(largura * 4, 512) * altura \quad (36)$$

Isso é consequência do fato de que foram usados exclusivamente números de ponto flutuante de 32-bits (4 bytes), e a estrutura de dados utilizada, *cudaSurfaceObject*, apresenta memória alinhada à 512 bytes. Isso significa que para uma imagem de 512x512 pixels, o programa consome aproximadamente 11,5 MB da memória, o que corresponde a aproximadamente 0,2% da capacidade da placa NVIDIA GTX 1060 6GB.

Para o processo de treinamento, necessitamos de estruturas de dados adicionais para armazenar dados referentes à imagem *gold-standard*, bem como imagens intermediárias necessários para aceleração do cálculo da Distância de Hausdorff e coeficiente de Dice, e uso do procedimento *reduce_by_key* (mencionado anteriormente na seção 5.7.1). Temos que o consumo de memória total é então descrito pela equação 37:

$$Total_{RCGA} \approx altura * (\max(largura, 512) + \max(largura * 4, 512) * (15 * N + 1)) \quad (37)$$

Sendo N o número de indivíduos em cada geração do algoritmo genético. No caso de imagens de 512x512 pixels e 30 indivíduos, temos que o processo de treinamento consome um pico de 473,2 MB de memória da placa de vídeo, correspondendo aproximadamente a 7,9% da capacidade da placa NVIDIA GTX 1060 6GB.

6.12. Análise em performance e complexidade computacional

6.12.1. Performance de uma segmentação individual

Para os ensaios de performance a seguir, 10 segmentações de 1000 iterações cada foram realizadas utilizando cada uma das três diferentes implementações do algoritmo DRLSE: A implementação de referência em MATLAB (Li, 2010b), a implementação serial em C++, e a implementação paralela em CUDA. Os tempos médios de execução, em função do tamanho da imagem, são exibidos na Tab. 14.

Tabela 14 – Comparação da performance para uma segmentação individual.

Tamanho da imagem	MATLAB (s)	C++ serial (s)	CUDA (s)	MATLAB / CUDA	C++ serial / CUDA
128x128	3,313 ± 0,048	14,465 ± 0,058	0,018 ± 0,001	184,1	803,6
256x256	8,655 ± 0,052	57,739 ± 0,274	0,044 ± 0,005	197,7	1312,3
512x512	70,540 ± 0,396	228,892 ± 2,254	0,147 ± 0,001	479,9	1557,1
1024x1024	274,390 ± 0,487	924,158 ± 5,376	0,547 ± 0,001	501,6	1690,0

Observando os resultados da Tab. 14, a vantagem em performance da implementação em CUDA sobre as implementações em MATLAB e em C++ serial é imediatamente aparente. Especialmente para o caso de uma imagem de 1024x1024 pixels, o programa em C++ demora em média 1700 vezes a duração do programa em CUDA para completar 1000 iterações, enquanto que o programa em MATLAB demora aproximadamente 500 vezes a duração do programa em CUDA. Deve-se notar, porém, que a implementação em MATLAB serve apenas como uma referência, e não foi desenvolvida necessariamente com performance em mente.

É também possível notar como que o programa em C++ é quase que perfeitamente linear em complexidade com relação ao número de pixels, ou seja, apresenta complexidade $O(largura * altura)$, como se esperaria de um programa puramente serial em uma única *thread*. O mesmo não pode ser observado nos dados coletados da implementação de referência em MATLAB. MATLAB em si deve utilizar de mecanismos de paralelização, como SIMD.

Os dados referentes à implementação em CUDA sugerem uma complexidade em tempo superior ao linear. Isso pode ser observado também na última coluna, onde a razão entre o tempo ocupado pelo programa em C++ e o tempo ocupado pelo programa em CUDA aumenta conforme o tamanho da imagem aumenta.

A imagem da Fig. 50 mostra uma captura de tela da interface do programa NVVP ao analisar a execução de um treinamento com o programa desenvolvido em CUDA. Note que o processo de análise realizado pelo NVVP significativamente distorce os tempos registrados, de modo que tal figura não deve ser usada como métrica de performance em si, mas apenas como indicativo da atividade da GPU durante a execução.

Figura 50 – Captura de tela do programa NVVP durante a análise da execução de uma segmentação com o programa desenvolvido em CUDA. A: Intervalo de 1000 iterações de uma segmentação com algoritmo DRLSE. B: Intervalo entre duas segmentações em CUDA.

É possível notar à esquerda da Fig. 50 como o LevelSetKernel ocupa aproximadamente 58% do tempo de execução, seguido pelo NormalizedGradientKernel

com 24,1%, e LaplaceKernel com 16,9%, sendo o restante ocupado por outros *kernels* menos relevantes. As respectivas linhas indicam a atividade de cada *kernel*, sendo azul/verde um período em que o *kernel* estava sendo executado, e uma lacuna branca um período em que não estava. É possível notar como não existem lacunas significativas entre os períodos de execução dos diversos *kernels*, e como a execução destes ocorre de modo paralelo com as tarefas de mover memória entre GPU e CPU, indicado com *cudaMemcpy* em laranja na terceira linha. Ambos os fatos indicam bom aproveitamento dos recursos do hardware.

Finalmente, o Intervalo delimitado em A indica as 1000 iterações de uma única execução de uma segmentação com algoritmo DRLSE.

6.12.2. Performance do processo de treinamento

A Tab. 15 indica o resultado da análise de performance do processo de treinamento dos parâmetros do algoritmo DRLSE, utilizando os algoritmos genéticos. Para cada teste, foram realizados 10 treinamentos com 60 gerações e 30 indivíduos cada, com e sem o uso do cache (discutido na seção 5.6). A duração média para o treinamento, junto com o respectivo desvio-padrão, foi registrada. O cache foi reinicializado antes de cada treinamento, mesmo quando ativo. Em todos os casos, as imagens utilizadas foram redimensionadas para tamanho de 512x512 pixels, e foi descartado o tempo de inicialização e cópia de memória para a GPU.

Tabela 15 – Tempo de execução de um treinamento com 30 indivíduos e 60 gerações.

Cache	Tempo de execução por treinamento (s)
Ativado	207,3013 $\pm$ 4,3778
Desativado	283,3567 $\pm$ 0,4306

Note que normalmente o processo de treinamento termina caso seja atingido um limiar para a métrica combinada Q_S , configurado por padrão em 0,99. Para tais ensaios, esse mecanismo de interrupção foi desabilitado, de modo a retratar o limite superior de duração do processo de treinamento.

Os cálculos realizados em CPU referentes ao processamento dos indivíduos (aplicação dos operadores genéticos, ordenação de acordo com *fitness*, entre outros) são realizados em série com as execuções e validações das segmentações (em GPU), mas apresentam impacto em performance desprezível. Isso pode ser observado na Fig. 50, sendo indicado em B a lacuna que representa o período entre diferentes gerações durante o processo de treinamento. A linha azul/verde onde tal lacuna se encontra indica a atividade da GPU de um modo geral, de modo que a lacuna em si corresponde ao único período em que a GPU se encontra em espera, entre iterações. Visualizado com NVVP, tal lacuna corresponde a um intervalo em tempo de aproximadamente $500\ \mu s$, correspondendo aproximadamente a 0,3% da duração total de uma segmentação. Levando-se em consideração a magnitude de tais impactos em performance, não foram tomadas medidas para se otimizar a parte em CPU da implementação do algoritmo DRLSE.

Adicionalmente, notou-se que o impacto em performance e memória do AG foi desprezível, primariamente por ser executado sempre com no máximo algumas dezenas de indivíduos. Desse modo, não foram tomadas medidas notáveis para otimizar a implementação do AG, e o impacto em performance da escolha de metaparâmetros para o AG não foi abordado neste trabalho.

7. CONCLUSÕES E EXTENSÕES

Observando os resultados da seção anterior, nota-se que a implementação com a plataforma CUDA permite uma drástica redução do tempo necessário para execução do algoritmo DRLSE, uma de suas principais desvantagens. Para uma imagem de 1024x1024 pixels, a implementação é capaz de realizar 1000 iterações do algoritmo em menos de um segundo em hardware acessível, trazendo o uso de tal algoritmo ao regime interativo.

O sistema de treinamento desenvolvido foi capaz de produzir conjuntos de parâmetros para o algoritmo DRLSE que acarretem em segmentações satisfatórias em todos os contextos de segmentação em que foi testado. Este sistema permite então a aplicação do algoritmo DRLSE sem conhecimento por trás de sua teoria, ou da interpretação e significado dos seus diversos parâmetros, cumprindo com parte dos objetivos iniciais do projeto.

Infelizmente, a alta sensibilidade do algoritmo a tais parâmetros faz com que seja necessário certo conhecimento de algumas regras relacionadas ao processo de treinamento. Por exemplo: existe a necessidade de que o contorno inicial se encontre à uma distância do objeto alvo comparável à distância entre contorno inicial e contorno *gold-standard* durante o processo de treinamento, observação esta discutida na seção anterior. Pode-se julgar que tais regras são mais acessíveis do que o conhecimento específico do funcionamento do algoritmo DRLSE, podendo-se argumentar que o efeito como um todo foi de simplificar o seu uso.

De um modo geral, conclui-se que os algoritmos genéticos funcionam bem para a otimização dos parâmetros do algoritmo DRLSE. Apesar disso, a sensibilidade do algoritmo DRLSE ao contexto de segmentação faz com que ainda seja necessária investigação adicional sobre o sistema desenvolvido como um todo, visando o uso pleno e robusto das capacidades de treinamento. No estágio atual, a flexibilidade do sistema desenvolvido é restrita, de modo que apesar de ser possível reutilizar conjuntos de parâmetros treinados em diferentes contextos, como inicialmente proposto, as condições de aplicação precisam ser similares às de treinamento para obtenção de resultados

satisfatórios. A elaboração da estratégia de treinamento escolhida, bem como a escolha de uma diferente abordagem de treinamento, pode conferir uma maior flexibilidade ao sistema.

Adicionalmente, a proposta inicial do sistema pode sugerir que seja possível encontrar um conjunto de parâmetros que evolua um contorno inicial a um contorno de referência, mas isso não é necessariamente possível. Isso não é uma falha do sistema desenvolvido, mas apenas uma limitação das capacidades de qualquer algoritmo de segmentação de imagens.

7.1. Publicações

Ao longo do desenvolvimento deste projeto, o artigo “Parallel Implementation of the DRLSE Algorithm” (COELHO, 2020) foi publicado ao congresso ICIAR, 2020. Tal artigo documenta a implementação em CUDA do algoritmo DRLSE, desenvolvida para este projeto.

O código fonte para COELHO, 2020, bem como eventualmente o código fonte do sistema desenvolvido para esta dissertação como um todo, estão disponíveis na página do autor dessa dissertação na plataforma Github: github.com/1danielcoelho.

7.2. Trabalhos futuros

O atual trabalho pode ser estendido em diversas maneiras. Pode-se por exemplo buscar otimizar ainda mais os tempos de aprendizado e segmentação, com a posterior otimização dos *kernels* usados. Apesar de que de modo geral a combinação de *kernels* acarrete em um ganho de performance, o programa em si é limitado pelo número de registros usados pelo LevelSetKernel. Potencialmente, separar o LevelSetKernel em *kernels* menores que usem menos registros pode levar a maior *occupancy* e melhor performance. Outro ganho em performance seria providenciado pela implementação de

alguma abordagem de otimização do algoritmo DRLSE, como a implementação em faixa estreita, descrita por Li, 2006.

O algoritmo de segmentação DRLSE foi sempre usado ao longo deste projeto com um número fixo de 1000 iterações. Como discutido na seção 6.3, os parâmetros disponíveis conferem ao processo de aprendizado graus de liberdade análogos à configuração do número de iterações. Apesar disso, um critério de parada mais preciso poderia ser determinado diretamente a partir da convergência do algoritmo DRLSE, por exemplo por meio do uso da funcional de sua formulação. A implementação de um critério de parada mais elaborado de modo paralelo não seria trivial, mas potencialmente providenciaria melhores segmentação, além de ganho em performance para o processo de treinamento de um modo geral.

Uma outra possibilidade de aprimoramento do sistema seria suporte para séries de imagens e segmentações em 3D. Isso pode ser feito com a expansão do sistema desenvolvido de modo a segmentar uma série de imagens com um único conjunto de parâmetros e contorno inicial bidimensional. Tal abordagem necessitaria, porém, de geração programática de contornos iniciais nas demais imagens da série. Uma outra abordagem possível para segmentação de objetos 3D seria a conversão do sistema inteiro ao domínio 3D, mas tal alteração seria acompanhada da maior dificuldade de delineação de superfícies iniciais e *gold-standard*, ao invés de simples contornos bidimensionais.

Um outro melhoramento importante seria a expansão do número de contextos de segmentação suportado pelo processo de treinamento. Atualmente, o algoritmo genético otimiza um único contexto, sendo então passível de sobre-ajuste: O conjunto de parâmetros produzido pode não ser genérico o suficiente para produzir bons resultados em contextos similares, sendo aplicável exclusivamente ao contexto de treinamento (resultado discutido na seção 6.9). Tal efeito pode ser mitigado fazendo com que o algoritmo de treinamento produza um conjunto de parâmetros que maximize a métrica de avaliação em diversos contextos de segmentação simultaneamente, convergindo a um conjunto de parâmetros mais genérico e robusto.

REFERÊNCIAS

ALTENBERG, L. *The Schema Theorem and Price's Theorem*. Foundations of Genetic Algorithms, Elsevier, v. 3, p. 23-49, 1995.

ANSÓTEGUI, C.; SELLMAN, M.; TIERNEY, K. *A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms*. In: Gent I.P. (eds) Principles and Practice of Constraint Programming - CP 2009. Lecture Notes in Computer Science, v. 5732, p. 142-157, 2009.

APPIA, V.; YEZZI, A. *Active Geodesics: Region-based Active Contour Segmentation with a Global Edge-based Constraint*. Proc IEEE Int Conf Comput Vis, v. 2011, p. 1975-1980, 2011.

BÄCK, T. *Evolutionary Algorithms in Theory and Practice*. Oxford: Oxford University Press, 1996.

BAKER, J. E. *Adaptive selection methods for Genetic Algorithms*. Proceedings of the International Conference on Genetic Algorithm and Their Applications, p. 101-110, Pittsburgh, PA, 1985.

BAKER, J. E. *Reducing bias and inefficiency in the selection algorithm*. Proceedings of the Second International Conference on Genetic Algorithms, p. 14-21, Cambridge, MA, 1987.

BALAPRAKASH, P.; BIRATTARI, M.; STÜTZLE, T. *Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement*. 4th International Workshop on Hybrid Metaheuristics, p. 108-122, 2007.

BALUJA, S.; CARUANA, R. *Removing the Genetics from the Standard Genetic Algorithm*. Proceedings of the Twelfth International Conference on Machine Learning. Tahoe City, California, USA, 1995.

BELL, N.; HOBEROCK, J. *Thrust: A Productivity-Oriented Library for CUDA*. GPU Computing Gems, Jade Edition, Edited by Wen-mei W. HWu, 2011. Disponível em: <<http://wnbell.com/blog/2011/10/01/thrust-productivity-oriented-library-for-cuda/>>, Acesso em: 30 dec. 2019.

BEYER, H.; SCHWEFEL, H. *Evolution strategies – A comprehensive introduction*. Natural Computing, v. 1, n. 1, p. 3-52, 2002.

BIRATTARI, M. et al. *A racing algorithm for configuring metaheuristics*. Proceedings of the Genetic and Evolutionary Computation Conference, p. 11-18, 2002

BIRATTARI, M. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. Tese de doutorado, Universite Libre de Bruxelles, Brussels, Belgium, 2004.

BLICKLE, T.; THIELE, L. *A Comparison of Selection Schemes used in Genetic Algorithms*. Zurich: Swiss Federal Institute of Technology. TIK-Report n. 11, v. 2, e. 2, 1995.

BLELLOCH, G. E.; MAGGS, B. M. *Parallel algorithms*. Em Algorithms and Theory of Computation Handbook. Chapman and Hall/CRC, v. 2, p. 25.

BOURD, A. *The OpenCL Specification*. Khronos OpenCL Working Group, v. 2.2-3, 2017.

BUSHBERG, J. T.; SEIBERT, J. A.; LEIDHOLDT JR, E. M.; BOONE, J. M. *The Essential Physics of Medical Imaging*. Philadelphia, PA, USA: Lippincott Williams & Wilkins, e. 3, 2012.

CANTÚ-PAZ, E. *Efficient and accurate parallel genetic algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

CASELLES, V. et al. *A geometric model for active contours*. Numerische Mathematik, v. 66, p. 1-31, 1993.

CHAN, T.; VESE, L. *Active Contours without edges*. IEEE Transactions on Image Processing, v. 10, n. 2, p. 266-277, 2001a.

CHAN, T.; VESE, L. *A level set algorithm for minimizing the Mumford-Shah functional in image processing*. IEEE Computer Society, Proceeding of the First IEEE Workshop on Variational and Level Set Methods in Computer Vision, p. 161-168, 2001b.

CHENG, J.; GROSSMAN, M.; MCKERCHER, T. *Professional CUDA C Programming*. John Wiley & Sons, Inc, Indianapolis, IN, USA, 2014.

CHOPP, D. L. *Computing Minimal Surfaces via Level Set Curvature Flow*. Journal of Comput. Physics, v. 106, n. 1, p. 77-91, 1993.

CLARK, K. et al. *The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository*, Journal of Digital Imaging, v. 26, n. 6, p. 1045-1057, 2013.

COELHO, D. P.; FURUIE, S. S. *Parallel Implementation of the DRLSE Algorithm*. ICIAR 2020, LNCS 12132, p. 25-35, doi: 10.1007/978-3-030-50516-5_3, 2020.

CPTAC - National Cancer Institute Clinical Proteomic Tumor Analysis Consortium. *Radiology Data from the Clinical Proteomic Tumor Analysis Consortium Cutaneous Melanoma (CPTAC-CM) Collection (Data set)*. The Cancer Imaging Archive, 2018a. Disponível em: <<https://doi.org/10.7937/k9/tcia.2018.odu24gze>>, Acesso em: 05 oct. 2019.

CPTAC - National Cancer Institute Clinical Proteomic Tumor Analysis Consortium. *Radiology Data from the Clinical Proteomic Tumor Analysis Consortium Pancreatic Ductal Adenocarcinoma (CPTAC-PDA) Collection (Data set)*. The Cancer Imaging Archive, 2018b. Disponível em: <<https://doi.org/10.7937/k9/tcia.2018.sc20fo18>>, Acesso em: 05 oct. 2019.

DAVIS, L. D. *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

DICE, L. R. *Measures of the amount of ecologic association between species*. Ecology, v. 26, n. 3, p. 297-302, 1945.

DE JONG, K. A. *An Analysis of the Behaviour of a class of Genetic Adaptive Systems*. Doctoral Dissertation, Dissertation Abstracts International, 36, 10, 5140B, University Microfilms No 76-9381, 1975.

DEB, K.; AGRAWAL, R. B. *Simulated Binary Crossover for Continuous Search Space*. Complex Systems, v. 9, n. 2, p. 115-148, 1995.

DEB, K.; BEYER, H. *Self-Adaptive Genetic Algorithms with Simulated Binary Crossover*. Evolutionary Computation, v. 9, n. 2, p. 197-221, 2001.

DEB, K.; KARTHIK, S.; OKABE, T. *Self-Adaptive Simulated Binary Crossover for Real-Parameter Optimization*. Proceedings of the 9th annual conference on Genetic and Evolutionary Computation (GECCO-2007), UCL London, p. 1187-1194, 2007.

DEBATTISTI, S.; MARLAT, N.; MUSSI, L.; CAGNONI, S. *Implementation of a Simple Genetic Algorithm within the CUDA Architecture*. GPUs for Genetic and Evolutionary Computation Competition at 2009 Genetic and Evolutionary Computation Conference (GECCO), 2009.

DLUGOSZ, F.; ROSOT, N. C.; ROSOT, M. A. D.; OLIVEIRA, Y. M. M. *Índice para a Avaliação de Segmentação de Imagens*. Floresta, Curitiba, v. 39, n. 1, p. 131-143, 2009.

EIBEN, A. E. et al. *Genetic algorithms with multi-parent recombination*. Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature, p. 78-87, 1994.

EKLUND, A.; DUFORT, P.; FORSBERG, D.; LACONTE, S. M. *Medical image processing on the GPU – Past, present and future*. Medical Image Analysis, v. 17, p. 1073-1094, 2013.

ERICKSON, B. J. et al. *Radiology Data from The Cancer Genome Atlas Liver Hepatocellular Carcinoma [TCGA-LIHC] collection*. The Cancer Imaging Archive, 2016. Disponível em: <<http://doi.org/10.7937/K9/TCIA.2016.IMMQW8UQ>>, Acesso em: 01-jan-2020.

ERICKSON, B. *Data From LGG-1p19qDeletion*. The Cancer Imaging Archive, 2017. Disponível em <[10.7937/K9/TCIA.2017.dwehtz9v](http://doi.org/10.7937/K9/TCIA.2017.dwehtz9v)>, Acesso em: 01 jan. 2010.

ESHELMAN, L. J.; SCHAFFER, D. *Real-Coded Genetic Algorithms and Interval-Schemata*. Foundations of Genetic Algorithms, Elsevier, v. 2, p. 187-202, 1993.

FENSTER, A.; CHIU, B. *Evaluation of Segmentation Algorithms for Medical Imaging*. Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference, Shanghai, China, 1-4 set. 2005.

GOLDBERG, D. E. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, Reading, 1989.

GOLDBERG, D. E. *Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking*. Complex Systems, v. 5, n. 2, 1991a.

GOLDBERG, D. E.; KALYANMOY D. *A Comparative Analysis of Selection Schemes Used in Genetic Algorithms*. Foundations of Genetic Algorithms, v. 1, p. 69-93, 1991b.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. e. 2, Prentice-Hall, Inc., Upper Saddle River, NJ, 2002.

GRATCH, J.; DEJONG, G. *Composer: A probabilistic solution to the utility problem in speed-up learning*. Proceedings of the Tenth National Conference on Artificial Intelligence, p. 235–240, 1992.

GREFENSTETTE, J. J. *Optimization of control parameters for genetic algorithms*. IEEE Trans. on Systems, Man, and Cybernetics, v. 16, p. 128, Janeiro 1986.

GREFENSTETTE, J. J.; BAKER, J. E. *How genetic algorithms work: A critical look at implicit parallelism*. San Mateo, CA: Morgan Kaufmann Publishers. Proceedings of the Third International Conference on Genetic Algorithms, p. 20-27, 1989.

GUPTA, A. et al. *PCSeg: Color Model Driven Probabilistic Multiphase Level Set based Tool for Plasma Cell Segmentation in Multiple Myeloma*. Aceito para PLOSONe Journal, 2018.

GUPTA, A. et al. *GCTI-SN: Geometry-Inspired Chemical and Tissue Invariant Stain Normalization of Microscopic Medical Images*. [S.I.], 2018.

GUPTA, R.; GUPTA, A. *MiMM_SBILab Dataset: Microscopic Images of Multiple Myeloma*. The Cancer Imaging Archive, 2019. Disponível em: <<https://doi.org/10.7937/tcia.2019.pnn6aypl>>, Acesso em: 05 oct. 2019.

GUPTA, R. et al. *Stain Color Normalization and Segmentation of Plasma Cells in Microscopic Images as a Prelude to Development of Computer Assisted Automated Disease Diagnostic Tool in Multiple Myeloma*. 16th International Myeloma Workshop (IMW), India, March 2017.

HAUPT, R. L.; HAUPT S. E. *Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors*. Applied Computational Electromagn. Soc. J., v. 15, n. 2, p. 94-102, 2000.

HERLIHY, M. *The art of Multiprocessor Programming*. São Francisco: Morgan Kaufmann, Revised reprint, p. 536, 2012.

HINTERDING, R. *Gaussian mutation and self-adaptation for numeric genetic algorithms*. Proceedings of the 1995 IEEE International Conference on Evolutionary Computation, p. 384-389, 1995.

HOLLAND, J. H. *Adaptation in natural and artificial systems. An introductory analysis with application to biology, control and artificial intelligence*. University of Michigan Press: Ann Arbor, MI, EUA, 1975.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. *ParamILS: An Automatic Algorithm Configuration Framework*. Journal of Artificial Intelligence Research, v. 36, p. 267-306, 2009.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. *Sequential Model-Based Optimization for General Algorithm Configuration*. LION, p. 507-523, 2011.

HUTTER, F.; HOOS, H. H.; STÜTZLE, T. *Automatic Algorithm Configuration based on Local Search*. Proceedings of the 22nd national conference on Artificial Intelligence, p. 1152-1157, 2007.

KASS, M.; WITKIN, A.; TERZOPOULOS, D. *Snakes: Active Contour Models*. International Journal of Computer Vision, v.1, p. 321-333, 1988.

KIRKPATRICK, S.; GELATT, C. D.; Vecchi, M. P. *Optimization by Simulated Annealing*. Science, New Series, v. 220, n. 4598, p. 671-680, 1983.

KOENIG, A. C. *A study of mutation methods for evolutionary algorithms*. Advanced Topics on Artificial Intelligence, p. 1-8, 2002.

KRISHNAMURTHY, A.; MCMAINS, S.; HANNIEL, I. *GPU-accelerated Hausdorff distance computation between dynamic deformable NURBS surfaces*. Computer-Aided Design, v. 43, p. 1370-1379, 2011.

LI, C.; XU, C.; GUI, C.; FOX, M. D.; *Distance Regularized Level Set Evolution and Its Application to Image Segmentation*. IEEE Transactions on Image Processing, v. 19, n. 12, p. 3243-3254, 2010a.

LI, C.; XU, C.; KONWAR, K. M.; FOX, M. D. *Fast Distance Preserving Level Set Evolution for Medical Image Segmentation*. 9th International Conference on Control, Automation, Robotics and Vision, p. 1-7, 2006.

LI, C. *Full domain DRLSE implementation*. 2010b. Disponível em <<http://www.imagecomputing.org/~cmli/DRLSE/>>, Acesso em: 21 jun. 2017.

MAIER, A.; STEIDL, S.; CHRISTLEIN, V.; HORNEGGER, J. *Medical Imaging Systems: An Introductory Guide*. Springer International Publishing, v. 11111, 2018.

MATEO, P. M.; ALBERTO, I. *A mutation operator based on a Pareto ranking for multiple-objective evolutionary algorithms*. Journal of Heuristics, v. 18, n. 1, p. 53-89, 2012.

MICHALEWICZ, Z. *Evolutionary computation techniques for nonlinear programming problems*. International Transactions in Operational Research, v. 1, n. 1, p. 223-240, 1994.

MICHALEWICZ, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer, e. 3, 1996.

MILLER, B. L.; GOLDBERG, D. E. *Genetic Algorithms, Tournament Selection, and the Effects of Noise*. Complex Systems, v. 9, p. 193-212, 1995.

MINTON, S. *An analytic learning system for specializing heuristics*. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, p. 922-929, 1993.

MOHANTA, R. K.; SETHI, B. *A Review of Genetic Algorithm application for Image Segmentation*. Int. J. Computer Technology & Applications, v. 3, n. 2, p. 720-723, 2012.

MÜHLENBEIN, H.; SCHLIERKAMP-VOOSEN, D. *Predictive models for the breeder genetic algorithm*. Evolutionary Computation, v. 1, n. 1, 1993.

MUMFORD, D.; SHAH, J. *Optimal approximation by piecewise smooth functions and associated variational problems*. Comm. Pure Appl. Math., v. 42, p. 577-685, 1989.

MUSTRA, M.; DELAC, K.; GRGIC, M. *Overview of the DICOM Standard*. 50th International Symposium ELMAR, Zadar, Croatia, p. 39-44, 2008.

NEMA PS3 / ISO 12052, *Digital Imaging and Communications in Medicine (DICOM) Standard*. National Electrical Manufacturers Association, Rosslyn, VA, USA. Disponível em <<http://medical.nema.org/>>, Acesso em: 04 abr. 2019.

NOROUZI, A. et al. *Medical Image Segmentation Methods, Algorithms and Applications*. IETE Technical Review, v. 31, n. 3, p. 199-213, 2014.

NVIDIA Corporation. *Compute Unified Device Architecture Programming Guide*. NVIDIA: Santa Clara, v. 9.1, 2017.

NVIDIA Corporation. *Profiler user's guide*, v. 10.1, 2019.

NYÚL, L. G.; FALCÃO, A. X.; UDUPA, J. K. *Fuzzy-connected 3D image segmentation at interactive speeds*. Graphical Models, v. 64, p. 259-281, 2003.

ORTIZ-BOYER, D.; HERVÁS-MARTÍNEZ, C.; GARCÍA-PEDRAJAS, N. *Improving crossover operator for real-coded genetic algorithms using virtual parents*. Journal of Heuristics, v. 13, n. 3, p. 265-314, 2007.

OSHER, S.; SETHIAN, J. A. *Fronts propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations*. J. Comput. Phys, v. 79, p. 12-49, 1988.

POLI, R. *Recursive Conditional Schema Theorem, Convergence and Population Sizing in Genetic Algorithms*. Proceedings of the Foundations of Genetic Algorithms Workshop (FOGA 6), p. 146-163, 2000.

POLI, R.; MCPHEE, N. F. *General schema theory for genetic programming with subtree swapping crossover: Part I*. Evolutionary Computation, v. 11, n. 2, p. 169-206, 2003.

POSPICHAL, P.; JAROS, J.; SCHWARZ, J. *Parallel Genetic Algorithm on the CUDA Architecture*. EvoApplications 2010, Part I, LNCS 6024, p. 442-451, 2010.

RADCLIFFE, N. J. *Equivalence class analysis of genetic algorithms*. Complex Systems, v. 2, n. 5, p. 183-205, 1991.

ROTH, H. R. et al. *A new 2.5 D representation for lymph node detection in CT*. The Cancer Imaging Archive, 2015. Disponível em <<http://doi.org/10.7937/K9/TCIA.2015.AQIIDCNM>>. Acesso em: 31 dec. 2019.

ROTH, H. R. et al. *A new 2.5 D representation for lymph node detection using random sets of deep convolutional neural network observations*. Medical Image Computing and Computer-Assisted Intervention, MICCAI 2014, p. 520-527, 2014.

RUOY, E.; TOURIN, A. *A viscosity solutions approach to shape-from-shading*. SIAM J. of Numerical Analysis, v. 23, n. 3, p. 867-888, 1992.

SEFF, A. et al. *2D view aggregation for lymph node detection using a shallow hierarchy of linear classifiers*. Medical Image Computing and Computer-Assisted Intervention, MICCAI 2014, p. 544-552, 2014.

SHEN X.; SURI J.S.; LAXMINARAYAN S. *Basics of PDEs and Level Sets*, 2002.

SUETENS, P. *Fundamentals of Medical Imaging*. New York, USA: Cambridge University Press, e. 2, 2009.

SURI J.S.; LAXMINARAYAN S. *PDE and Level Sets: Algorithmic Approaches to Static and Motion Imagery*. Topics in Biomedical Engineering. Springer, Boston, MA, 2002.

TAN, H. et al. *A Novel Level Set Model for Image Segmentation with Interactive Label Regularization Term*. Computer Vision. CCCV 2017. Communications in Computer and Information Science, v. 773, 2017.

TAHA, A. A.; HANBURY, A. *Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool*. BMC medical imaging, v. 15, n. 29, 2015.

TING, C-K. *On the Mean Convergence Time of Multi-parent Genetic Algorithms Without Selection*. Advances in Artificial Life, p. 403-412, 2005.

TOWNSEND, A. *Genetic Algorithms – A Tutorial*, p. 1-52, 2013

UDUPA, J. K.; SAHA, P. K.; LOTUFO, R. A. *Relative Fuzzy Connectedness and Object Definition: Theory, Algorithms, and Applications in Image Segmentation*. IEEE Transactions On Pattern Analysis And Machine Intelligence, v. 24, n. 11, p. 1485-1500, 2002.

UDUPA, J.K. et al. *A framework for evaluating image segmentation algorithms*. Comput Med Imaging Graph. v. 30, n. 2, p. 75-87, 2006.

VISHKIN, U. *Thinking in Parallel: Some Basic Data-Parallel Algorithms and Techniques*. College Park, UMD, 2010. Disponível em <<http://www.umiacs.umd.edu/users/vishkin/PUBLICATIONS/classnotes.pdf>>, Acesso em: 05 oct. 2019.

WANG, X. et al. *Enhanced distance regularization for re-initialization free level set evolution with application to image segmentation*. Neurocomputing, v. 141, p. 223-235, 2014.

WHITE, D. *An overview of Schema Theory*. arXiv preprint: 1401.2651, 2014. Disponível em: <<https://arxiv.org/abs/1401.2651>>, Acesso em: 25 sep. 2019.

WHITLEY, D. *A genetic algorithm tutorial*. Statistics and Computing, n. 2, v. 4, p. 65-85, 1994.

WRIGHT, A. *Genetic Algorithms for Real Parameter Optimization*. Foundations of Genetic Algorithms 1. Elsevier, v. 1, p. 205-218, 1991.

XU, M. H.; LIU, Y. Q.; HUANG, Q. L.; ZHANG, Y. X.; LUAN, G. F. *An improved Dijkstra's shortest path algorithm for sparse network*. Applied Mathematics and Computation, v. 185, e. 1, p. 247-254, 2007.

ZEYNETTIN, A. et al. *Predicting Deletion of Chromosomal Arms 1p/19q in Low-Grade Gliomas from MR Images Using Machine Intelligence*. J Digit Imaging, v. 30, n. 4, p. 469-476, 2017.

ZHANG, K.; ZHANG, H.; SONG, H.; ZHANG, D. *Reinitialization-free level set evolution via reaction diffusion*. IEEE Trans Image Process, v. 22, n. 1, p. 258-271, 2013.

ZHUGE, Y.; CAO, Y.; MILLER, R. W. *GPU Accelerated Fuzzy Connected Image Segmentation by using CUDA*. Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, p. 6341-6344, 2009.