IMPLEMENTAÇÃO DO ALGORITMO DE TREINAMENTO DO CLASSIFICADOR FLORESTA DE CAMINHOS ÓTIMOS EM GPU

IWASHITA, Adriana Sayuri
Universidade Estadual Paulista (UNESP Bauru/SP)
sayuri@fc.unesp.br

PAPA, João Paulo
Universidade Estadual Paulista (UNESP Bauru/SP)
papa@fc.unesp.br

RESUMO: Técnicas de reconhecimento de padrões tem como principal objetivo classificar um conjunto de amostras, sendo o processo de aprendizado a fase de maior consumo de tempo. O problema pode piorar em ferramentas de classificação interativas, o que pode ser inaceitável para grandes bases de dados. Um exemplo de classificador é o baseado em Floresta de Caminhos Ótimos [8] - OPF. Dado que muitos trabalhos tem sido orientados à implementação de algoritmos de reconhecimento de padrões em ambiente General Purpose Graphics Processing Unit - GPGPU, o presente estudo objetivou a implementação da etapa de treinamento do classificador Floresta de Caminhos Ótimos em CUDA, visando aumentar a sua eficiência. A otimização do classificador em CUDA demonstrou uma fase de treinamento mais rápida que a versão original.
PALAVRAS-CHAVE: Reconhecimento de padrões. Floresta de Caminhos Ótimos. GPU.

ABSTRACT: *Pattern recognition techniques have as main objective to classify a set of samples, in which the learning process is the most time-consuming phase. The problem may become worse in interactive classification tools, and may be unacceptable for large databases. An example of a classifier is the one based on Optimum-Path Forest – OPF [8]. Since many studies have been oriented to the implementation of pattern recognition algorithms on General Purpose Graphics Processing Unit - GPGPU environment, this study aimed the implementation of the training stage of the Optimum-Path Forest classifier in CUDA, in order to increase its efficiency. The classifier optimization in CUDA has shown a training phase faster than the original version.*
KEYWORDS: *Pattern Recognition. Optimum-Path Forest. GPU.*

INTRODUÇÂO

Técnicas de reconhecimento de padrões tem como principal objetivo classificar um conjunto de amostras (padrões) baseadas em um conhecimento a priori ou em alguma informação estatística obtida dessas amostras [12]. Os métodos de classificação podem ser divididos em não-supervisionados, supervisionados ou semi-supervisionados de acordo com seus algoritmos de aprendizado, sendo que o aprendizado de um classificador é o que mais consome esforço computacional. Um classificador baseado em floresta de caminhos ótimos (Optimum-Path Forest - OPF) foi apresentado por Papa et. al. [8], com duas variantes propostas: (i) a versão não supervisionada [9] e a (ii) versão supervisionada, a qual é subdividida em OPF com grafo completo [8] e OPF com grafo k-vizinhos mais próximos (k-Nearest Neighbors - k-NN) [7], sendo a que faz uso do grafo completo a mais utilizada e difundida.

Ultimamente, empresas produtoras de placas de vídeo, tais como NVIDIA e ATI, empenharam-se em produzir placas gráficas com GPUs (Graphic Processor Unit) cada vez mais poderosas, ocasionando em um crescente aumento na programação orientada aos processadores gráficos, os quais são projetados especialmente para executar operações em paralelo, possibilitando, assim, um notável aumento no desempenho quando comparados às CPUs. Operações habituais em um microproces-

sador comum, tais como operações com pontos flutuantes, por exemplo, podem ser executadas e, principalmente, programadas agora em GPUs com mais facilidade do que antigamente.

Recentemente, diversos trabalhos orientados à implementação de técnicas de reconhecimento de padrões em GPU foram desenvolvidos, tais como implementações de redes neurais e SVMs orientadas aos processadores gráficos. A motivação para a implementação dessas abordagens em GPU reside no fato de elas possuírem uma fase de treinamento muito custosa. Nesse contexto, o presente trabalho objetivou o estudo do algoritmo de treinamento supervisionado do classificador OPF e do uso da GPU para programação de propósito geral, visando a implementação do algoritmo de treinamento do OPF no ambiente CUDA para execução em GPUs, para que no futuro seja construída a cudaOPF, uma biblioteca de desenvolvimento de classificadores de padrões baseados em floresta de caminhos ótimos em ambiente CUDA.

1. RECONHECIMENTO POR FLORESTA DE CAMINHOS ÓTIMOS

Seja *Z* uma base de dados e Z_1 e Z_2 os conjuntos de treinamento e teste, respectivamente, com $|Z_1|$ e $|Z_2|$ amostras, tais que $Z = Z_1 \cup Z_2$. Seja, também, $S \in Z_1$ um conjunto de protótipos de todas as classes (isto é, amostras importantes que melhor representam as classes). A distância *d(s,t)* entre duas amostras, *s* e *t,* é dada pela distância entre seus vetores de características. Pode-se utilizar qualquer métrica válida (Euclidiana, por exemplo) ou algum algoritmo de distância mais elaborado [15]. Seja (Z_1,A) um grafo completo cujos nós são as amostras em Z_1, onde qualquer par de amostras define um arco em *A* (isto é, $A = Z_1 \times Z_1$) (Figura 1a), gerando um grafo completo e ponderado do conjunto Z_1. O algoritmo OPF pode ser utilizado com qualquer função de custo suave que pode agrupar amostras com propriedades similares [1]. Entretanto, o OPF foi projetado usando a função de custo f_{max}, por causa de suas propriedades teóricas para estimar protótipos ótimos [2]:

$$f_{max}(\langle s \rangle) = \begin{cases} 0 & se\ s \in S, \\ +\infty & caso\ contrário \end{cases}$$

$$f_{max}(\pi \cdot \langle s,t \rangle) = max\{f_{max}(\pi), d(s,t)\} \quad (1)$$

sendo que $f_{max}(\pi)$ computa a distância máxima entre amostras adjacentes em (π) quando (π) não é um caminho trivial.

O algoritmo OPF associa um caminho ótimo *P*(s)* de *S* a toda amostra $s \in Z_1$, formando uma floresta de caminhos ótimos *P* (uma função sem ciclos, a qual associa a todo $s \in Z_1$ seu predecessor *P(s)* em *P*(s),* ou uma marca *NIL* quando $s \in S$. Seja *R(s)* $\in S$ a raiz de P*(s), a qual pode ser alcançada usando P(s). O algoritmo OPF computa, para cada $s \in Z_1$, o custo *V(s)* de *P*(s)*, o rótulo $L(s)=\lambda(R(s))$ e o seu predecessor P(s).

A Figura 1b ilustra a Floresta de caminhos ótimos resultante de Figura 1a para e dois protótipos (nós contornados). As entradas (x,y) acima dos nós denotam, respectivamente, o custo e o rótulo das amostras. Os arcos direcionados indicam os nós predecessores no caminho ótimo. Na fase de classificação (Figura 1c), é calculado para a amostra de teste (triângulo branco) as suas conexões (linhas tracejadas) com os nós de treinamento. Como resultado da classificação (Figura 1d), o rótulo 2 e o custo de classificação 0.3 é associado à amostra de teste, valores obtidos através do caminho ótimo do protótipo mais fortemente conexo.

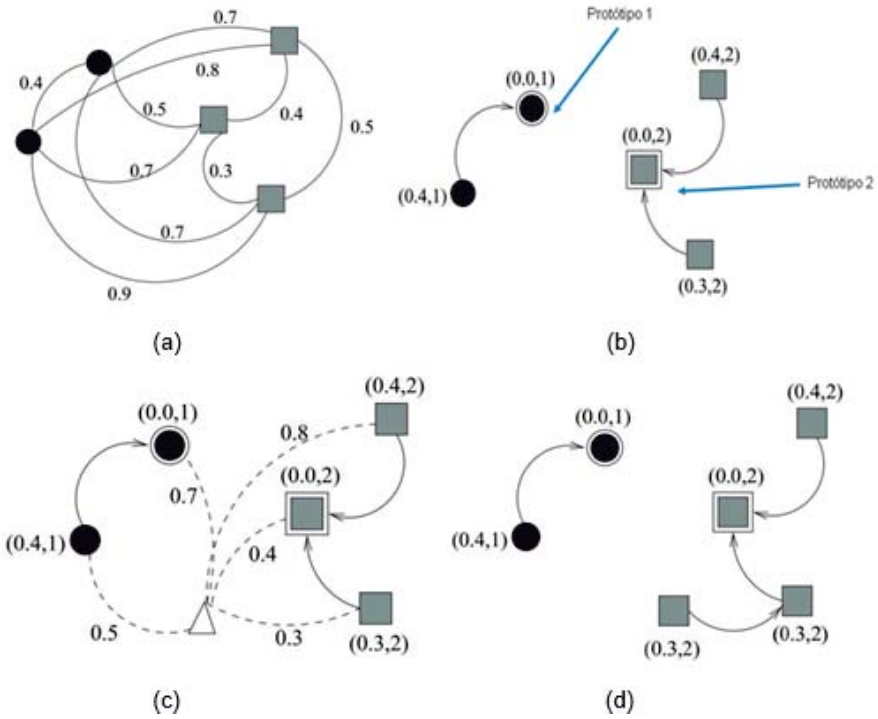


Figura 1: (a) Grafo completo ponderado. (b) Floresta de caminhos ótimos resultante para e dois protótipos. (c) Amostra de teste e suas conexões com os nós de treinamento. (d) Caminho ótimo do protótipo mais fortemente conexo.

1.1. Treinamento

*S** é um conjunto ótimo de protótipos quando o algoritmo propaga os rótulos $L(s)=\lambda(s)$ para todo $s \in Z_1$. Desta forma, *S** pode ser encontrado explorando a relação teórica entre a Árvore de Espalhamento Mínima (*Minimum Spanning Tree* - MST) [2] e a árvore de caminhos mínimos para f_{max} . O treinamento consiste essencialmente em duas etapas: encontrar *S** e um classificador OPF enraizado em *S**. Computando uma MST no grafo completo (Z_1,A), obtém-se um grafo conexo acíclico cujos nós são todas as amostras em Z_1, e os arcos são não direcionados e ponderados. Seus pesos são dados pela distância *d* entre os vetores de características de amostras adjacentes. Esta árvore de espalhamento é ótima no sentido em que a soma dos pesos de seus arcos é mínima se comparada a outras árvores de espalhamento no grafo completo. Na MST, cada par de amostras é conectado por um caminho, o qual é ótimo de acordo com . Os protótipos ótimos são os elementos conectados na MST com diferentes rótulos em Z_1, isto é, elementos mais próximos de classes diferentes. Removendo-se os arcos entre classes diferentes, tais amostras adjacentes tornam-se protótipos em *S** e pode-se computar uma floresta de caminhos ótimos sem erros de classificação em Z_1 (processo de competição). Portanto, cada amostra protótipo define sua própria árvore de caminhos ótimos (*Optimum-Path Tree* - OPT), e a coleção de todas as OPTs definem a floresta de caminhos ótimos (OPF), que dá nome ao classificador.

1.2. Classificação

A classificação de um elemento *t* ocorre após a etapa de treinamento, sendo que para qualquer amostra $t \in Z_2$, consideramos todos os arcos que conectam t com as amostras $s \in Z_1$, como se t fizesse parte do grafo de treinamento. Considerando todos os caminhos possíveis de *S** a *t*, encontramos o caminho ótimo $P^*(t)$ de *S** e rótulo *t* com a classe $\lambda(R(t))$ do seu protótipo mais

fortemente conectado $R(t) \in S^*$. Este caminho pode ser identificado incrementalmente por meio da avaliação do custo ótimo $C(t)$ como:

$$C(t) = min\{max\{C(s), d(s,t)\}\}, \forall s \in Z_1 \qquad (2)$$

Seja o nó $s^* \in Z_1$ o único que satisfaz a Equação 2 (ou seja, o predecessor $P(t)$ no caminho ótimo $P^*(t)$). Dado que $L(s^*)=\lambda(R(t))$, a classificação simplesmente atribui $L(s^*)$ como a classe de t. Um erro ocorre quando $L(s^*) \neq \lambda(t)$.

2. OTIMIZAÇÃO DA ETAPA DE TREINAMENTO

A implementação do algoritmo de treinamento supervisionado do classificador OPF no ambiente CUDA foi realizado em determinadas instruções do código para que executasse ações semelhantes em paralelo. Como as placas gráficas tem sua estrutura de processamento organizado em arquiteturas paralelas, os algoritmos neles implementados devem ter comportamentos paralelizáveis para que ocorra ganho de

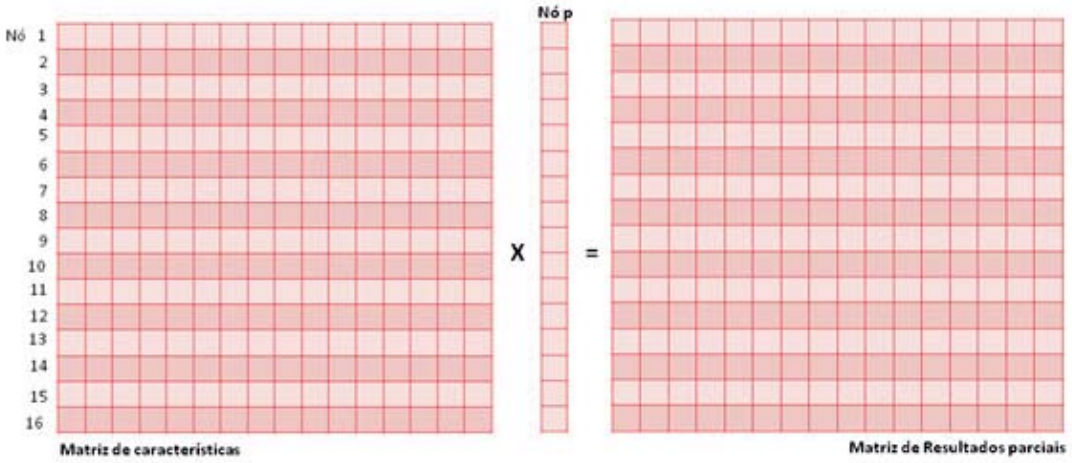


Figura 2: Cálculo das distâncias do nó p.

desempenho. Conforme descrito na Seção 2.1., a etapa de criação do OPF é composto por duas fases: (i) calculo da MST sobre o grafo completo e marcação dos protótipos que são os elementos conectados com diferentes rótulos, e (ii) com os protótipos, iniciar o processo de conquista entre eles para construir as OPTs. A Figura 1 ilustra este procedimento. De forma semelhante, a abordagem proposta do classificador OPF em GPU paraleliza a criação da MST e o processo de competição entre protótipos. Para ambas as abordagens, há três etapas principais: encontrar o elemento p com menor custo que vai tentar conquistar os outros nós, calcular a distância entre o nó p e os demais nós, e verificar se o nó p conquista outros nós ou não. Na criação da MST, será associado no inicio um custo 0 a raiz r e custo +∞ nos demais nós para controle da criação da MST. O próximo passo é, dentre os nós do grafo, encontrar o menor deles, que será o líder, ou seja, a amostra p com menor custo. Para encontrar o elemento de menor custo, utilizou-se a primitiva min_element da biblioteca de algoritmos paralelos Thrust [6].

O próximo passo consiste em calcular a distância entre o nó p e os nós restantes. Neste caso, utilizou-se um conjunto de threads para cuidar de um conjunto de nós. Isto significa que a distância entre o nó p e os demais nós do grafo serão processados em paralelo, utilizando uma abordagem semelhante à multiplicação de uma matriz por um vetor [10]. As linhas desta matriz, chamada de “Matriz de características” (Figura 2), são os nós do conjunto de treinamento representados por seus vetores de características. Esta matriz é alocada na memória de textura, e pode ser acessado por todas as threads; o vetor de características do nó p é alocado na memória global. Uma thread é alocada para cada elemento

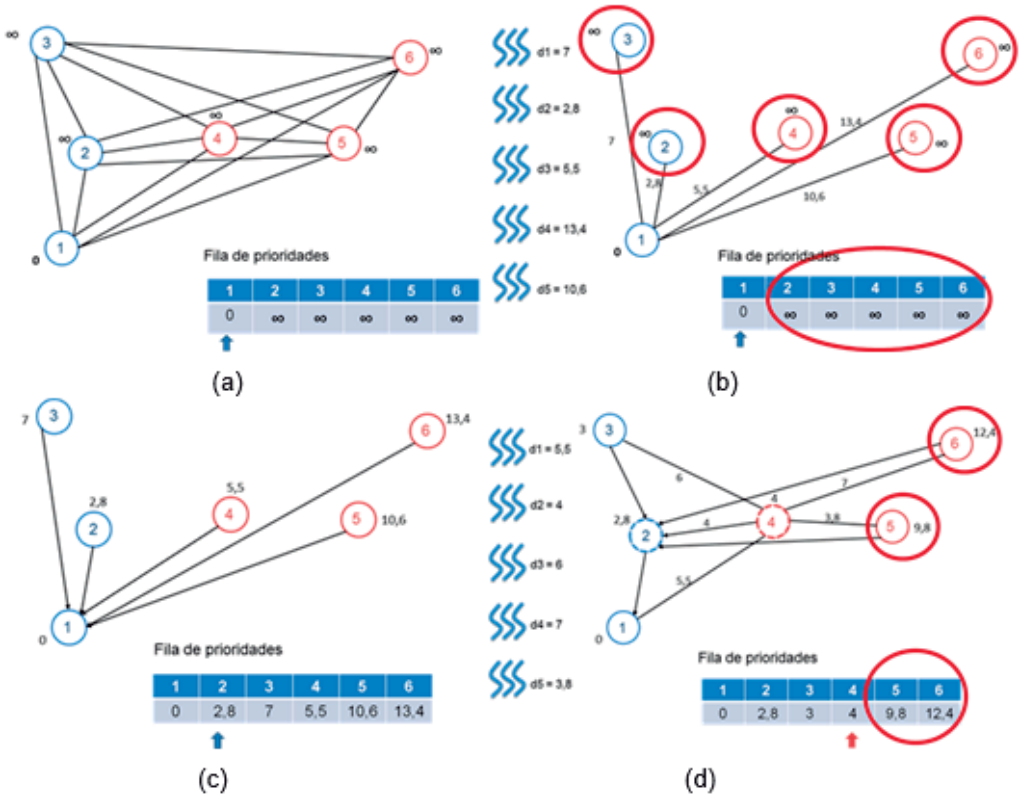


Figura 3: (a) Grafo completo. (b) Nó 1 conquista demais nós, cada conjunto de threads calcula uma distância diferente. (c) Atualiza custos e nó 2 é o próximo a sair da fila. (d) Após 4ª iteração, dois protótipos são encontrados e nó 4 conquista dois nós.

da Matriz de características, que irá calcular a distância entre um elemento da matriz e um elemento do vetor do nó p, armazenando esse resultado em uma "Matriz de resultados parciais". Cada bloco de threads tem uma Matriz de resultados parciais, que é alocada em sua memória compartilhada. O resultado final (as distâncias entre p e demais nós) é um vetor de distâncias alocado na memória global, na qual a posição i é a distância entre o nó *p* e o nó *i.* O nível de paralelismo dessas operações pode ser classificado como granularidade fina, pois há um grande número de processamentos pequenos e simples executando em nível de instrução.

A Figura 3a ilustra o conjunto de nós a ser treinado: os nós do grafo são inicializados com custo +∞ e o nó *1* será a raiz da MST, sendo portanto atribuído a ele o custo zero. O primeiro passo do algoritmo de treinamento do OPF é encontrar os protótipos, o que pode ser feito computando a MST e então marcando os nós mais próximos de classes diferentes. A Figura 3b ilustra a situação do conjunto de dados quando está na primeira iteração. Com a primitiva paralela *min_element,* o nó *1* sai da fila para a conquista de nós. Cada conjunto de threads na GPU é responsável por calcular uma distância entre o nó 1 e os demais nós; também verifica se o custo do nó ao qual está "responsável'" (nó *i*) é menor que o valor da distância oferecido pelo nó *1.* A Figura 3c ilustra a situação do conjunto após a primeira iteração, e a escolha do nó *2* (usando primitiva *min_element*), para o próximo passo. Os nós pontilhados na Figura 3d representam os protótipos encontrados após a 4ª iteração e, como destacado pela seta, o nó *4* iniciará o processo de conquista. Do mesmo modo que na Figura 3b, o nó *4* sai da fila para a conquista de outros nós, cada conjunto de threads na GPU é responsável

por calcular uma distância entre o nó *4* e os demais nós; também verifica se o custo do nó i é menor que o valor da distância oferecido pelo nó *4*. A Figura 4 apresenta a MST gerada após a primeira fase do treinamento e os nós pontilhados definem os protótipos.

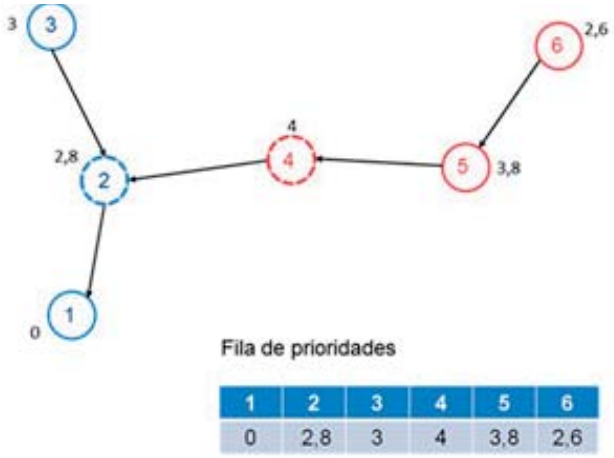


Figura 4: MST gerada, 2 protótipos encontrados.

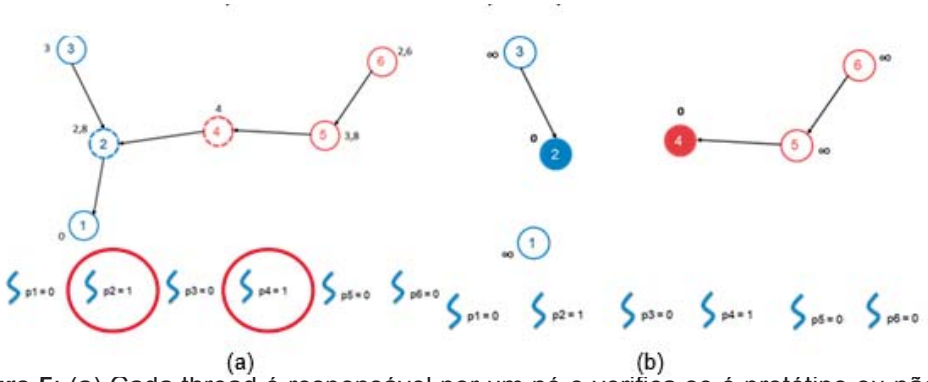


Figura 5: (a) Cada thread é responsável por um nó e verifica se é protótipo ou não-protótipo. (b) Cada thread atualiza o custo do nó e, caso este seja protótipo, atualiza também o predecessor.

A Figura 5a ilustra o conjunto de nós após a construção da MST. Nesta fase, cada *thread* é responsável por atualizar um nó: cada *thread* verifica se o nó foi definido como protótipo ou não. Caso o nó seja protótipo, a *thread* atualiza o custo do nó com o valor zero e atualiza o predecessor deste nó como *NIL*. Caso o nó não seja protótipo, o valor do custo será +∞. Já a Figura 5b ilustra o conjunto após a atualização dos nós.

A Figura 6a ilustra a segunda fase do treinamento, que é computar a OPF dado os protótipos encontrados: a primitiva *min_element* escolhe o nó de menor custo, no caso o nó protótipo 2, que irá sair da fila para a conquista de nós. A Figura 6b mostra a conquista do nó 2: cada conjunto de *threads* na GPU é responsável por calcular uma distância entre o nó 2 e os demais nós. Utilizando a função de custo f_{max}, o nó 2 tenta conquistar os demais nós. A Figura 7 apresenta a floresta de caminhos ótimos resultante.

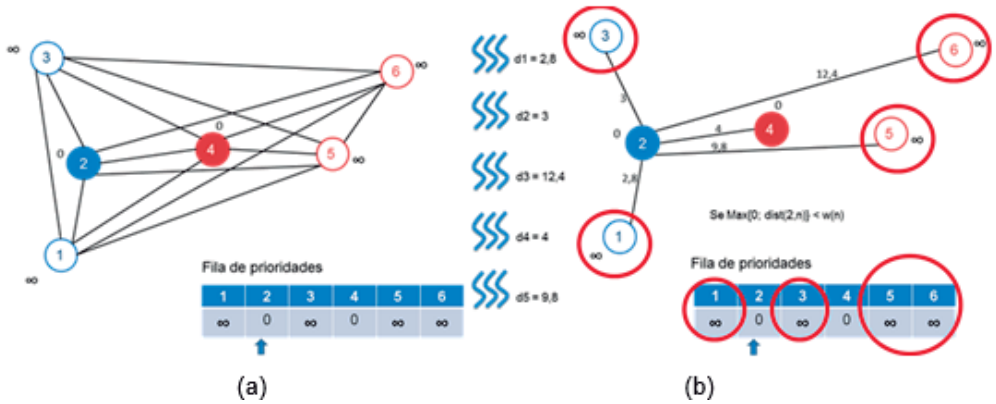


Figura 6: (a) Grafo com protótipos encontrados e custos atualizados. (b) Cada conjunto de threads calcula uma distância, nó protótipo 2 só não conquista o outro protótipo.

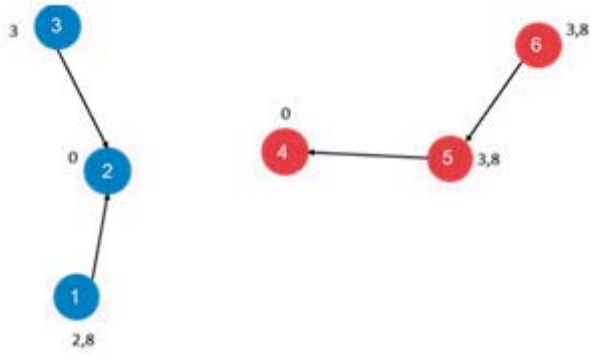


Figura 7: Floresta de caminhos ótimos resultante.

3. EXPERIMENTOS

Esta seção descreve os experimentos realizados para o treinamento em GPU/CUDA proposto. Foram utilizadas diversas bases de dados públicas com diferentes tamanhos para avaliar a eficiência do treinamento e também a acurácia da classificação em cenários distintos, conforme descreve a Tabela 1.O ambiente onde foi realizado os experimentos possui as seguintes características: placa mãe *Intel Corporation DX58SO (Smackover);* processador Intel *Core i7 - 960 - 3.20GHz com 8M Cache*; dois pentes de memória *4GB DDR3 1067MHz DIMM Kingston;* disco rígido *Seagate Barracuda 1TB - 7200RPM;* e duas placas de vídeo *NVIDIA*

Tabela 1 - Bases utilizadas nos experimentos.

Base	Nº Amostras	Nº Características	Nº Classes
Connect [3]	67557	126	3
Covtype (25\%) [3]	145250	54	7
IJCNN [5]	49990	22	2
Indian Pine [4]	21025	220	18
Letter [11]	15000	16	26
Mnist [14]	60000	780	10
Poker [3]	25010	10	10
Salinas [13]	111104	204	17

Corporation GF 110 (GeForce GTX 570). O sistema operacional utilizado foi o *Ubuntu 12.04 LTS - 64bit* com Kernel *Linux 3.2.0-25.* As especificações para o trabalho em CUDA são: *Cuda compilation tools release 4.2* com 480 *Cuda cores*; e a biblioteca de algoritmos paralelos *Thrust versão v1.5.1.*

A seguir, os resultados obtidos:As Figuras 8a e 8b apresentam, respectivamente, o tempo de treinamento e a média da acurácia para as diferentes porcentagens do conjunto de treinamento e teste para a base de dados Connect. Quando utilizamos o conjunto de treinamento com 30% da base Connect, a abordagem proposta teve um ganho de 2,523 para o treinamento. Já

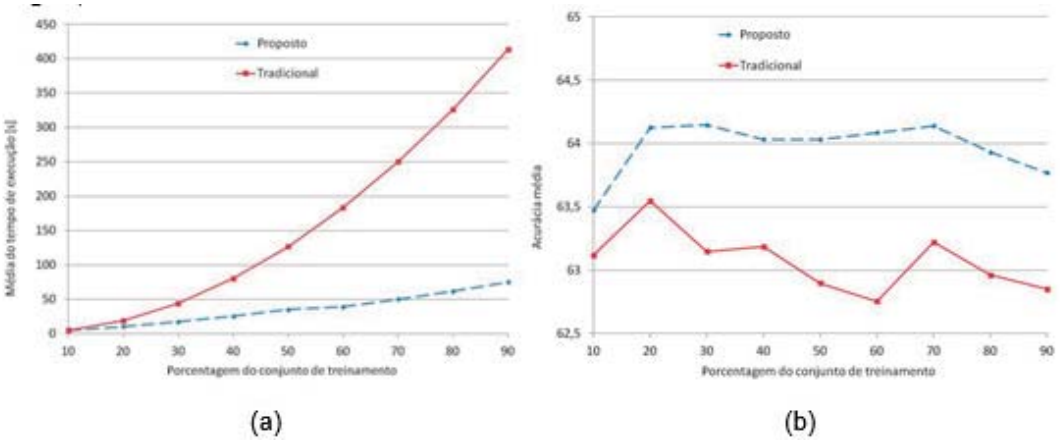


Figura 8: Resultados para conjunto de dados Connect. (a) Média de tempo de execução (em segundos) da etapa de treinamento. (b) Média da acurácia da classificação.

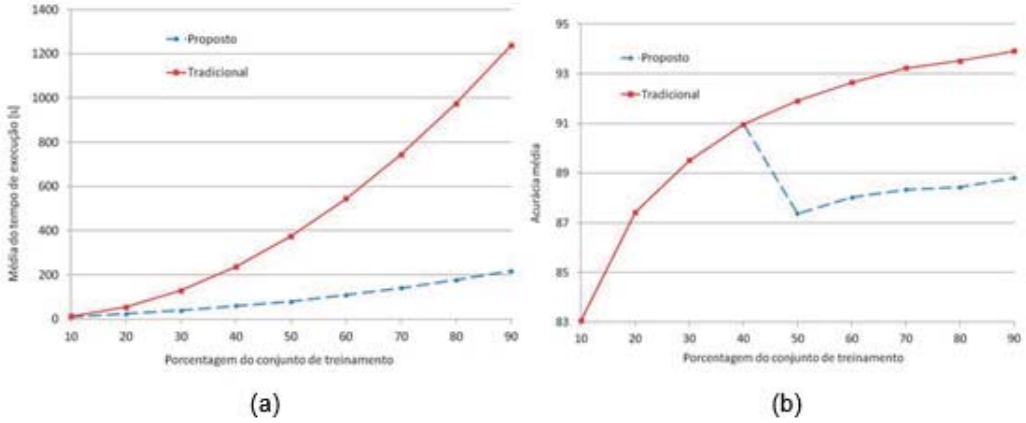


Figura 9: Resultados para 25% do conjunto de dados Covtype. (a) Média de tempo de execução (em segundos) da etapa de treinamento. (b) Média da acurácia da classificação.

para 90% de base de treinamento, a abordagem proposta foi 5,522 mais rápida.

As Figuras 9a e 9b apresentam o tempo de treinamento e a média da acurácia para 25% da base de dados Covtype. Quando utilizamos o conjunto de treinamento com 10% da base 25% Covtype, a abordagem proposta teve um ganho de 1,188 para o treinamento. Já para 90% de base de treinamento, a abordagem proposta foi 5,690 mais rápida.

As Figuras 10a e 10b apresentam o tempo de treinamento e a média da acurácia para a base de dados IJCNN. Quando utilizamos o conjunto de treinamento com 70% da base IJCNN, a abordagem proposta teve um ganho de 1,528 para o treinamento. Já para 90% de base de treinamento, a abordagem proposta foi 1,793 mais rápida.

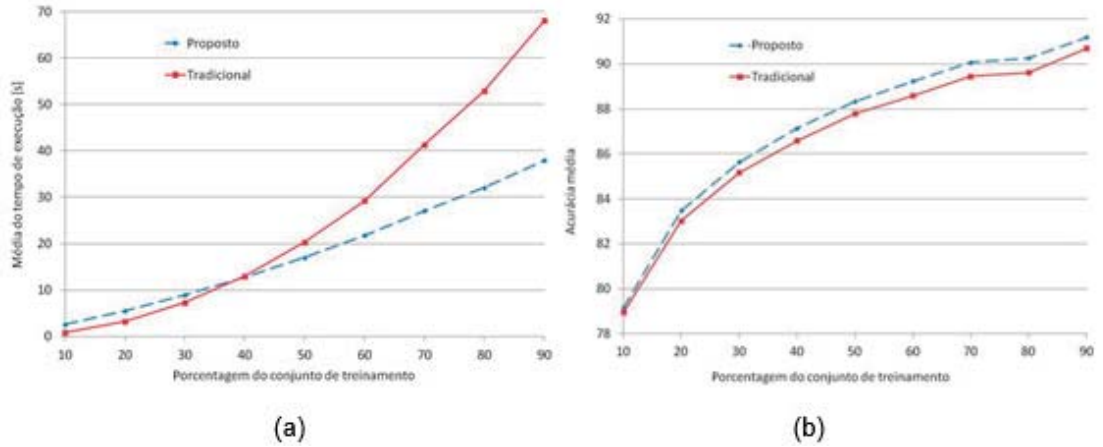


Figura 10: Resultados para conjunto de dados IJCNN. (a) Média de tempo de execução (em segundos) da etapa de treinamento. (b) Média da acurácia da classificação.

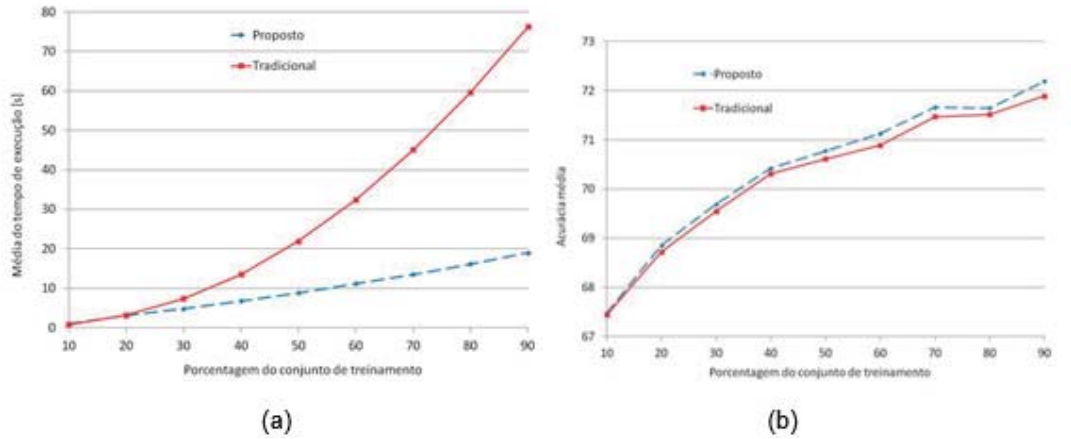


Figura 11: Resultados para conjunto de dados IJCNN. (a) Média de tempo de execução (em segundos) da etapa de treinamento. (b) Média da acurácia da classificação.

As Figuras 11a e 11b apresentam o tempo de treinamento e a média da acurácia para a base de dados Indian Pine. Quando utilizamos o conjunto de treinamento com 40% da base Indian Pine, a abordagem proposta teve um ganho de 2,004 para o treinamento. Já para 90% de base de treinamento, a abordagem proposta foi 4,021 mais rápida.

As Figuras 12a e 12b apresentam o tempo de treinamento e a média da acurácia para a base de dados Letter. Verificamos que, quando utilizamos o conjunto de treinamento com 90% de base de treinamento, a abordagem proposta foi 1,825 vezes mais lenta que a abordagem tradicional.

As Figuras 13a e 13b apresentam o tempo de treinamento e a média da acurácia para a base de dados Mnist. Quando utilizamos o conjunto de treinamento com 10% da base Mnist, a abordagem proposta teve um ganho de 5,711 para o treinamento. Já para 90% de base de treinamento, a abordagem proposta foi 14,405 mais rápida.

As Figuras 14a e 14b apresentam o tempo de treinamento e a média da acurácia para a base de dados Poker. Verificamos que, quando utilizamos o conjunto de treinamento com 90% de base de treinamento, a abordagem proposta foi 2,118 vezes mais lenta que a abordagem tradicional.

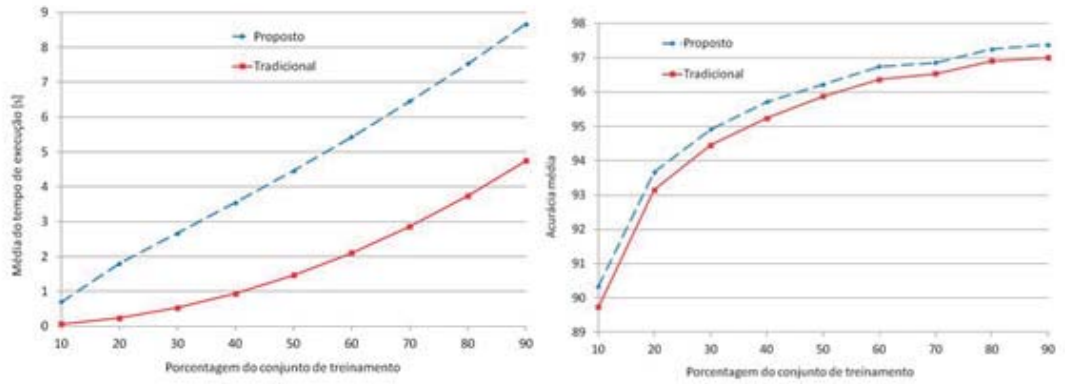


Figura 12: Resultados para conjunto de dados Letter. (a) Média de tempo de execução (em segundos) da etapa de treinamento. (b) Média da acurácia da classificação.

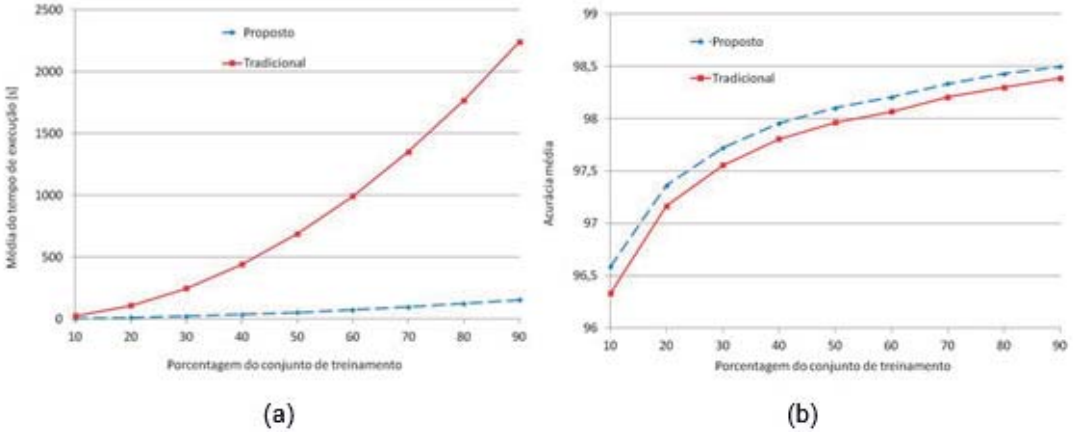


Figura 13: Resultados para conjunto de dados Mnist. (a) Média de tempo de execução (em segundos) da etapa de treinamento. (b) Média da acurácia da classificação.

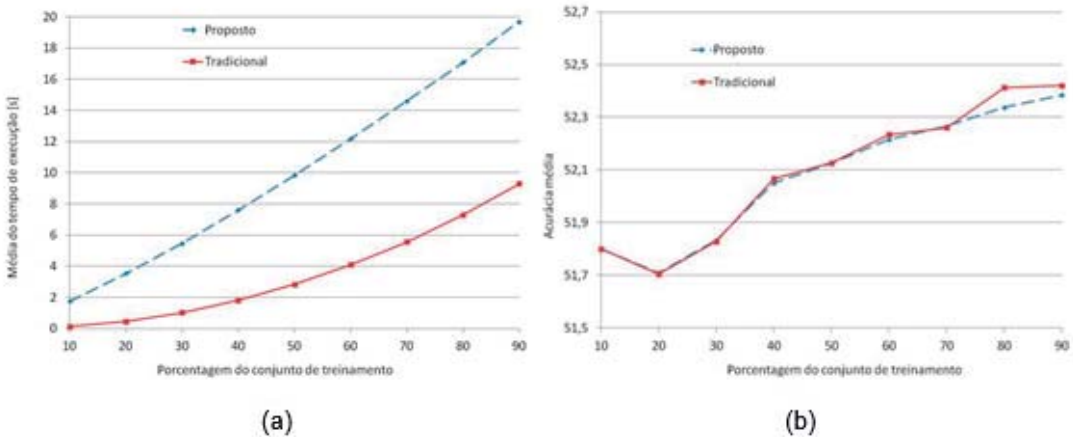
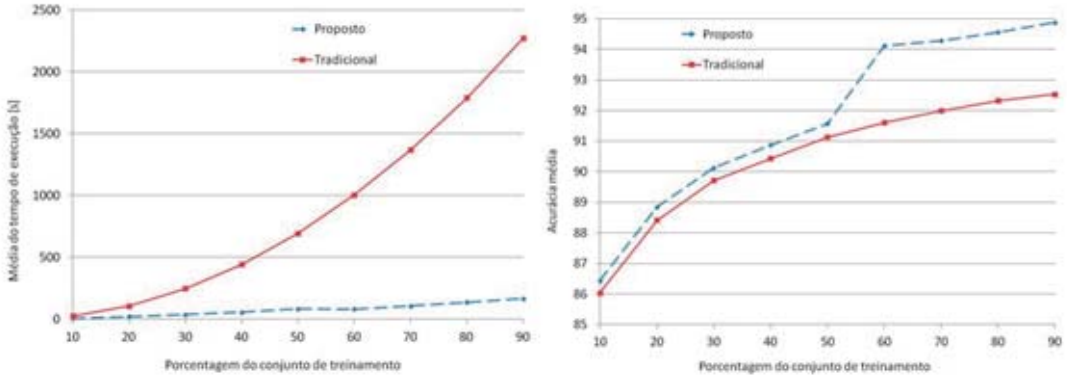


Figura 14: Resultados para conjunto de dados Poker. (a) Média de tempo de execução (em segundos) da etapa de treinamento. (b) Média da acurácia da classificação.



(a) (b)

Figura 15: Resultados para conjunto de dados Salinas. (a) Média de tempo de execução (em segundos) da etapa de treinamento. Média da acurácia da classificação.

CONSIDERAÇÕES FINAIS

A área de reconhecimento de padrões é essencial para a computação, podendo também esta ser empregada em diversos domínios de aplicação, tais como medicina, mineração de dados, automação industrial, reconhecimento biométrico e sensoriamento remoto. Entretanto, em situações nas quais a base de dados é muito grande, o custo do treinamento de um algoritmo de classificação pode não ser satisfatório e demandar muito tempo. Dado que ultimamente empresas de placas de vídeo tem investido cada vez mais em placas gráficas com GPUs, tem-se verificado que está cada vez mais simples e barata a utilização de tais componentes para programação paralela e vários autores já fazem uso desses ambientes.

Neste trabalho, foram estudadas técnicas de programação em ambientes paralelos, bem como o classificador Floresta de Caminhos Ótimos. A etapa de treinamento do algoritmo OPF em ambiente GPU-CUDA foi proposta utilizando a ideia de multiplicação de matriz por vetor de modo paralelo.

Resultados experimentais demonstraram que a técnica proposta obteve taxas de reconhecimento similares às obtidas pela abordagem tradicional, porém com uma etapa de treinamento mais rápida na maioria dos casos. Bases que possuem poucos elementos com um vetor de características pequeno não apresentam uma grande melhora no desempenho. Uma possível causa para o não ganho seria o custo que a CPU tem para alocar os dados das threads da memória da CPU para a GPU, ou seja, a sobrecarga devido ao carregamento de memória da CPU e GPU. Quando o vetor de características é maior, observamos um ganho de até 14 vezes em determinados casos.

Como trabalhos futuros, um próximo passo seria implementar outras versões do classificador OPF em ambiente CUDA, tais com o algoritmo de aprendizado do OPF supervisionado, bem como sua versão não supervisionada.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] A.X. Falcão, J. Stolfi, and R.A. Lotufo. The image foresting transform: Theory, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 26(1):19–29, 2004.

[2] C. Allène, J.Y. Audibert, M. Couprie, J. Cousty, and R. Keriven. Some links between min-cuts, optimal spanning forests and watersheds. In *Proceedings of the 8th International Symposium on Mathematical Morphology*, pages 253–264, 2007.

[3] Covertype. Covertype dataset. uci - machine learning repository. Available: *http://archive.ics.uci.edu/ml/datasets/Covertype*, 2012.

[4] D.A. Landgrebe. *Signal Theory Methods in Multispectral Remote Sensing*. Wiley, Newark, NJ, 2005.

[5] Danil Prokhorov. Ijcnn 2001 neural network com-

petition. Available: *http://www.geocities.com/ ijcnn/ nnc_ijcnn01.pdf*, 2001.

[6] J. Hoberock and N. Bell. Thrust: A parallel template library. Available: *http://www.meganewtons.com/,* 2010. Version 1.3.0.

[7] J. P. Papa and A. X. Falcão. A new variant of the optimum-path forest classifier. In *Proceeding of the 4th International Symposium on Advances in Visual Computing,* pages 935–944, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] J.P. Papa, A.X. Falcão, and C.T.N. Suzuki. Supervised pattern classification based on optimum-path forest. *International Journal of Imaging Systems and Technology,* 19(2):120–131, 2009.

[9] L.M. Rocha, F.A.M. Cappabianco, and A.X. Falcão. Data clustering as an optimum-path forest problem with applications in image analysis. *International Journal of Imaging Systems and Technology,* 19(2):50–68, 2009.

[10] N. Fujimoto. Faster matrix-vector multiplication on geforce 8800gtx. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, 2008.

[11] R. D. King, C. Feng, and A. Sutherland. Statlog: Comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence,* 9(3):289–333, 1995.

[12] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification (2nd Edition). *Wiley-Interscience*, 2000.

[13] S. Kaewpijit, J.L. Moigne, and T.A. El-Ghazawi. Automatic reduction of hyperspectral imagery using wavelet spectral analysis. *IEEE Transactions on Geoscience and Remote Sensing,* 41(4):863–871, 2003.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Available: *http://yann.lecun.com/exdb/mnist/,* November 1988.

[15] Y. P. Wang and T. Pavlidis. Optimal correspondence of string subsequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 12(11):1080–1087, 1990.

Adriana Sayuri Iwashita: Possui formação em Ensino médio e Técnico-profissionalizante em Informática pelo Colégio Técnico Industrial “Isaac Portal Roldán “ (2006), graduação em Bacharelado em Ciência da Computação pela Universidade Estadual Paulista “Júlio de Mesquita Filho” UNESP (2010) e mestrado em Ciência da Computação pela UNESP (2013). Atualmente é aluno especial de doutorado da Universidade Federal de São Carlos.

João Paulo Papa: Possui graduação em Bacharelado em Sistemas de Informação pela Universidade Estadual Paulista Júlio de Mesquita Filho (2002), mestrado em Ciência da Computação pela Universidade Federal de São Carlos (2005), doutorado em Ciência da Computação pela Universidade Estadual de Campinas (2008) e pós-doutorado em Ciência da Computação pela Universidade Estadual de Campinas (2009). Atualmente, é Professor Assistente Doutor no Departamento de Computação da Universidade Estadual Paulista Júlio de Mesquita Filho, Unesp-Bauru e pesquisador colaborador junto ao Instituto de Computação da Universidade Estatual de Campinas. Tem experiência na área de Ciência da Computação, com ênfase em Sistemas de Computação, atuando principalmente nos seguintes temas: processamento de imagens, aprendizado de máquina e reconhecimento de padrões.