ESTUDO COMPARATIVO DE METAHEURÍSTICAS PARA TREINAMENTO DE CLASSIFICADOR ADABOOST APLICADO À DETECÇÃO DE CÉLULAS SANGUÍNEAS

Trabalho de Conclusão de Curso

Engenharia da Computação

Willamos de Aguiar Siqueira e Silva
Orientador: Prof. Dr. Carmelo J. Albanez Bastos-Filho

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

Willamos de Aguiar Siqueira e Silva

ESTUDO COMPARATIVO DE METAHEURÍSTICAS PARA TREINAMENTO DE CLASSIFICADOR ADABOOST APLICADO À DETECÇÃO DE CÉLULAS SANGUÍNEAS

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, dezembro de 2012.

ESCOLA POLITÉCNICA
DE PERNAMBUCO

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 7 de 12 de 2012, às 16:00 horas, reuniu-se para deliberar a defesa da monografia de conclusão de curso do discente WILLAMOS DE AGUIAR SIQUEIRA E SILVA, orientado pelo professor Carmelo José Albanez Bastos Filho, sob título ESTUDO COMPARATIVO DE METAHEURÍSTICAS PARA TREINAMENTO DE CLASSIFICADOR ADABOOST APLICADO À DETECÇÃO DE CÉLULAS SANGUÍNEAS, a banca composta pelos professores:

Sérgio Campello Oliveira

Carmelo José Albanez Bastos Filho

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

☒ Aprovada ☐ Aprovada com Restrições* ☐ Reprovada

e foi-lhe atribuída nota: 9,0 (Nove)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O discente terá 07 dias para entrega da versão final da monografia a contar da data deste documento.

SÉRGIO CAMPELLO OLIVEIRA

CARMELO JOSÉ ALBANEZ BASTOS FILHO

* Este documento deverá ser encadernado juntamente com a monografia em versão final.

Agradecimentos

Gostaria primeiramente de agradecer aos meus pais. Sem eles não teria tido a educação que tive e exemplos de foco, responsabilidade e dedicação. Gostaria de agradecer também aos meus amigos mais próximos, que desde a infância me deram apoio para seguir atrás de meus objetivos. André Renato, Gustavo Ide, Renan Salvador, Dayvson Costa, Doron Reginatto, Cláudio Maciel, Ludmilla Albuquerque e Daianne Nobre, vocês fazem parte de mim.

Quero agradecer também aos meus demais parentes, em especial minhas avós Neide e Djanira, meus tios Wellington Siqueira, José Galdino, Antônio Aguiar e minha tia Giovanna Aguiar, meus primos Ítalo, Débora e Eliane, Rafael e Antônio Claudio por sempre estarem ao meu lado ao longo dos anos.

Gostaria também de agradecer a todos os meus professores por quem passei por todos esses anos, desde o jardim de infância até a faculdade. Em especial, gostaria de agradecer aos professores Mêuser Valença, Sérgio Campello, José Paulo, Bruno Fernandes, Sérgio Murilo, Byron Leite e ao meu orientador Carmelo Bastos Filho. Obrigado por terem contrubuído tanto para meu crescimento pessoal e acadêmico.

Também devo agradecer à minha namorada, Dayle Vasconcelos, por ter me dado tanta força e tanto ânimo para terminar este trabalho. Você tem feito da minha vida algo extremamente especial.

Por fim, gostaria de agradecer aos amigos que fiz dentro desta universidade: Alexandre Azevedo, Alessandra Maranhão, Bruno Yamashita, Rodrigo Augusto, Denise Assis, Lara Dantas, Bruna Galle, Breno Augusto, Marcos Gabriel, Débora Nascimento, Cristóvão Rufino, Igor Menezes, Eduardo Augusto, Jamisson Freitas e tantos outros. Vocês agregaram muito a minha caminhada.

Resumo

Este trabalho teve como objetivos comparar o desempenho entre técnicas de otimização para o treinamento do classificador *AdaBoost* para detecção de células sanguíneas. O foco foi dado na detecção de plaquetas por serem células sanguíneas mais simples, porém deve ser possível de ser extendido às demais células sanguíneas. *AdaBoost* é um classificador que tem como característica classificar padrões complexos a partir de padrões mais simples. Este classificador faz uso da técnica de *boosting*, onde um classificador forte é obtido através do cascateamento de classificadores fracos. Para otimizar o treinamento do classificador, foram testados os seguintes algoritmos: PSO (busca por enxame de partículas), FSS (busca por cardume) e GA (algoritmos genéticos). Foram realizadas trinta simulações utilizando dez mil iterações com trinta partículas, peixes ou indivíduos para efetuar a otimização do *AdaBoost* com cinco, dez e vinte classificadores fracos. Os resultados obtidos mostram que o GA gerou resultados mais robustos ao utilizar a taxa de acerto e taxa de falso positivo como métrica nos casos com cinco e dez classificadores, enquanto o PSO obteve melhores resultados para as mesmas métricas no caso com vinte classificadores.

Abstract

This work aimed to compare the performance of several optimization techniques for training the *AdaBoost* classifier in order detect blood cells. The focus was on platelets detection, since they are simpler, but it would be able to be extended to other blood cells. *AdaBoost* is able to classify complex patterns from simpler patterns. This classifier uses a technique called boosting which uses a weak classifiers cascade to compose a strong classifier. In order to optimize the *AdaBoost* training, the following algorithms were used: PSO (Particle Swarm Optimizartion), FSS (Fish School Search) and GA (Genetic Algorithms). Thirty executions were performed using ten thousand iterations with thirty particles, fishes or individuals to perform *AdaBoost* optimization, using five, ten or twenty weak classifiers. The results achieved shows that GA was able to find the best results for five and ten classifiers using hit rate and false positive rate as metrics, while PSO outperformed the other techniques for twenty classifiers using the same metrics.

Sumário

Conteúdo

Índice de Figuras

Tabela de Símbolos e Siglas

ABC – *Artificial Bee Colony* (Busca por colônia de abelhas)

DE - *Differential Evolution* (Evolução Diferencial)

FSS – *Fish School Search* (Busca por cardumes)

GA – *Genetic Algorithm* (Algoritmo Genético)

GB – Giga Bytes

GHz – Giga Hertz

PSO – *Particle Swarm Optimization* (Otimização por enxame de partícula)

RAM – Random-access memory (memória de acesso aleatório)

1. Introdução

Neste trabalho de conclusão de curso foram utilizadas técnicas de inteligência computacional para otimizar o treinamento de uma técnica de classificação, chamada *AdaBoost,* quando aplicado ao reconhecimento de células sanguíneas. Neste capítulo serão apresentados: a motivação na Seção 1.1; os objetivos gerais e específicos na Seção 1.2; e, por fim, a estrutura do restante do trabalho na Seção 1.3.

1.1. Motivação

A identificação e contagem de células sanguíneas é importante para que se possa automatizar o seu processo de. Neste trabalho de Conclusão de Curso (TCC), foi feito um estudo de como realizar a identificação de plaquetas, por serem células sanguíneas mais simples, porém espera-se que seja possível estender o trabalho para outros tipos de células sanguíneas.

Plaquetas são células sanguíneas produzidas na medula óssea em decorrência da fragmentação dos megacariócitos[1] [1] e têm como função principal a formação de um tampão responsável pela coagulação sanguínea. Efetuar a contagem destas células de maneira precisa é importante para detectar possíveis alterações, tanto quantitativas como qualitativas.

A diminuição da quantidade de plaquetas é chamada plaquetopenia ou trombocitopenia [2]. Plaquetopenia têm como principais causas leucemias agudas, AIDS e viroses em geral. Por outro lado, o aumento do número de plaquetas é chamado plaquetose ou trombocitose. Entre as principais causas da plaquetose pode-se citar leucemia mielóide crônica, períodos pós-hemorrágicos e doenças infecciosas crônicas.

[1] Megacariócitos são células da medula óssea responsáveis pela produção de plaquetas.

A contagem desse tipo de célula pode ser feita tanto de forma manual, utilizando o método de Fonio [2], por exemplo, onde se compara o número de plaquetas e o de eritrócitos, ou automatizada, que pode ser feita utilizando técnicas como dispersão de luz, tecnologia de impedância ou ainda uma combinação dessas duas técnicas.

Os métodos atualmente utilizados para contagem automatizada estão sujeitos a falhas. Aglomerados plaquetários ou plaquetas gigantes podem levar a um falso cenário de plaquetopenia. Por outro lado, a presença de fragmentação das hemácias pode resultar em um diagnóstico equivocado de plaquetose. Devido a problemas dessa natureza, a precisão na contagem de plaquetas atualmente tem baixa precisão, variando entre 22% e 66% [3].

Técnicas de reconhecimento de padrões podem ser aplicadas para o reconhecimento automatizado de plaquetas. Neste TCC, será utilizado um classificador chamado de *AdaBoost* [4], que tem como característica classificar padrões complexos a partir de padrões mais simples. Para otimizar o treinamento e aumentar a eficiência do *AdaBoost,* é possível fazer uso de técnicas de inteligência computacional [5]. Alguns algoritmos serão testados para o treinamento *AdaBoost* e será realizada uma comparação para verificar a adequabilidade de cada um ao problema. Pretende-se, portanto, melhorar um sistema capaz de classificar e contar plaquetas.

1.2. Objetivos

Técnicas de reconhecimento de padrões podem ser aplicadas para o reconhecimento automatizado de plaquetas. Neste TCC, foi utilizado o *AdaBoost*. Para otimizar o treinamento e aumentar a eficiência do *AdaBoost,* foram implementados tanto algoritmos evolucionários, quanto baseados em inteligência de enxames. Os algoritmos baseados em enxames implementados foram Otimização por enxames de partículas (PSO, *Particle Swarm Optimization*) [6] com topologia local e Busca por Cardumes (FSS, *Fish School Search*).*[7]* A técnica evolucionária utilizada foi Algoritmos Genéticos (GA, *Genetic Algorithm*) [6].

1.3. Estrutura do trabalho

Este trabalho possui 5 capútulos. O Capítulo 2 tem como objetivo fundamentar teoricamente este TCC. Nele serão abordados tópicos sobre o funcionamento do *AdaBoost* e das técnicas que serão utilizadas para otimizar o seu treinamento. No Capítulo 3 serão mostrados como os experimentos foram realizados e quais valores foram empregados para os parâmetros dos algoritmos. No Capítulo 4 serão apresentados os resultados obtidos. Por fim, o Capítulo 5 apresenta as conclusões e as perspectivas de trabalhos futuros.

2. Revisão Bibliográfica

Neste capítulo são abordados tópicos sobre o funcionamento do *AdaBoost* e das técnicas utilizadas para otimizar o seu treinamento. Nascimento [8] já mostrou que é possível utilizar o *AdaBoost* tendo seu treinamento otimizado pelo algoritmo PSO com sucesso para detecção de plaquetas.

2.1. AdaBoost

AdaBoost é um algoritmo desenvolvido por Viola e Jones [9] criado para realizar detecção de face de forma rápida e robusta. Esse algoritmo tem como base os conceitos de características de Haar e imagem integral [9].

2.1.1. Características de Haar

Uma característica de Haar é uma tentativa de representação de uma característica de uma dada imagem. Uma característica de Haar é composta por um retângulo com pelo menos uma região clara e uma ou mais regiões escuras. Figura 1mostra quatro possíveis características

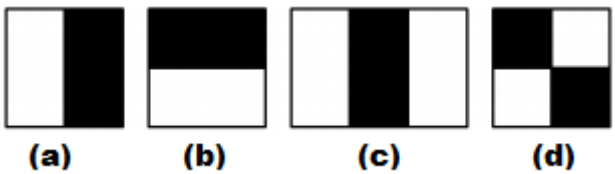

Figura 1 - Exemplo de características de Haar

A característica da Figura 1(a) é responsável por detectar bordas verticais, enquanto que a característica da Figura 1(b) é utilizada para detecção de bordas horizontais. A característica da Figura 1(c), por sua vez detecta linhas verticais, equanto que a última, Figura 1(d), é responsável por detectar linhas diagonais.

2.1.2. Imagem Integral

Imagem integral [4] consiste em uma forma de representar a informação contida em uma imagem. A representação da informação é feita pela soma da intensidade de todos os pixels à esquerda e à direita do pixel em questão. Sendo assim, o valor da informação associada ao ponto *(x,y)* pode ser obtida fazendo uso de $ii(x, y)$, que representa o valor da imagem integral no ponto *(x,y)*, e $i(x, y)$, que é a intensidade na imagem original

$$s(x, y) = s(x, y-1) + i(x, i), \tag{2.1}$$

$$i((x, y) = ii(x-1, y) + s(x, y), \tag{2.2}$$

em que $s(x, y)$ é a soma acumulada dos valores de uma linha, $s(x, y) = 0$ e $i((-1, y) = 0$.

Outra forma de calcular diretamente $ii((x, y)$ é fazendo uso da equação (2.3):

$$ii(x, y) = \sum_{x' \le x, y' \le y} i(x', y') . \tag{2.3}$$

Um exemplo de cálculo de imagem integral está apresentado na Figura 2. O valor do ponto 1 é o somatório das intensidades dos pixels contidos na subimagem *A*. O valor do ponto 2 é a dado por *A* + *B*. O valor na posição 3 é obtido pela soma *A*+*C*. Por fim, o valor do ponto 4 é dado pela soma das subjanelas A+B+C+D. Sendo assim, a soma para calcular apenas a sub-janela D em função dos pontos pode ser obtida através de “4+1–2–3”.

Utilizando a combinação dos conceitos de característica de Haar e de imagem integral, tem-se como calcular rapidamente o valor de cada característica fazendo o somatório dos pixels sobre a área escura do total acumulado, subtraído do total da intensidade dos pixels sobre a área da região clara de uma característica.

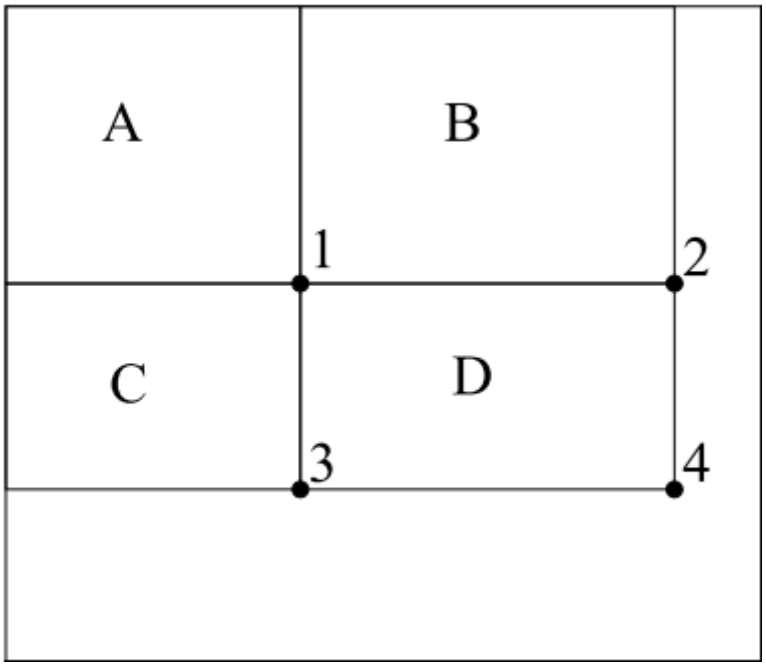

Figura 2 - Exemplo de imagem com suas subimagens *A, B, C* e *D*.

2.1.3. Treinamento *AdaBoost*

O *AdaBoost* baseia-se em em uma técnica chamada *boosting*, proposto por Freund e Scaphire [10]. Pode-se fazer uso dessa técnica para aumentar o desempenho de classificadores tidos como fracos para formarem uma cascata. Essa cascata de classificadores vai fazendo com que o erro seja diminuído a cada estágio [11].

Um classificador fraco $h_j(x)$ é uma estrutura composta por um vetor de características de Haar $f_j(x)$, um limiar θ_j e uma paridade p_j. O ponto chave do algoritmo é encontrar uma característica e um limiar associado a ela que separe melhor os exemplos positivos e negativos das imagens da base de dados. Para cada característica, deve ser encontrado o limiar ótimo para que o menor número possível de exemplos seja classificado de maneira incorreta. Um classificador fraco pode ser entendido conforme a equação (2.4):

$$h_j(x) = \begin{cases} 1, \textit{ caso } p_j f_j(x) < p_j \theta_j \\ 0, \textit{ caso contrário} \end{cases} \quad (2.4)$$

.

Uma das formas de efetuar o cálculo do limiar é seguindo a abordagem desenvolvida por Carbonetto [12]. Nessa abordagem, o limiar considera a distribuição de valores associados às características tanto dos exemplos positivos, quanto negativos. O cálculo pode ser realizado usando a equação (2.5), que por sua vez considera o valor da característica $f_j(x)$ para um dado exemplo apresentado e os conjutos de exemplos positivos C_0 e negativos C_1,

$$\theta_j(x) = 0{,}5\left(\frac{1}{|C_0|} \sum_{x \in C_0} f_j(x) + \frac{1}{|C_0|} \sum_{x \in C_1} f_j(x) \right) \quad (2.5)$$

.

A polaridade é atribuída conforme os exemplos são avaliados. Caso os exemplos positivos sejam calculados abaixo do limiar, é atribuída à polaridade o valor ‘1’. Caso contrário, é atribuída à polaridade o valor ‘-1’.

Como já foi mencionado anteriormente, o *AdaBoost* é um algoritmo de *boosting* que faz uso de uma cascata de classificadores fracos para gerar um classificador forte. Sendo assim, o classificador forte pode ser entendido como uma combinação dos classificadores anteriores, conforme mostrado na equação (2.6),

$$H(x) = \alpha_1 h_1 + \alpha_2 h_2 + \ldots + \alpha_n h_n \quad (2.6)$$

.

O treinamento do *AdaBoost* é feito de maneira supervisionada. As imagens da base de treinamento são apresentadas com rótulos de positivas ou negativas. O pseudo-código do *AdaBoost* está descrito no Algoritmo 1.

A cascata de classificadores funciona utilizando a noção de estágios, onde cada classificador fraco é um estágio. Dado um conjunto de imagens a ser analisado, uma imagem é considerada válida caso passe com sucesso por todos os estágios da cascata. Caso a imagem seja rejeitada em algum dos classificadores, ela é descartada e tida como falsa e não prossegue para os outros estágios. Este processo está exemplificado na Figura 3.

Uma vantagem de utilizar uma cascata de classificadores é poder rejeitar uma imagem sem ter que passar por todo o processo de classificação, como já foi citado anteriormente. Isso torna a avaliação como um todo mais rápida. A importância dessa velocidade é que, em uma imagem, a maior parte das subimagens divididas em busca de classificação forma uma imagem negativa para a classe que está sendo treinada.

Na cascata de classificadores, é interessante no início da cascata classificadores que eliminem o maior número de exemplos negativos possível e tenham um menor custo computacional. Cada camada subseqüente torna-se mais restritiva e segue eliminando exemplos com resposta negativa à classe que está sendo treinada. Sendo assim, a cascata pode ter um funcionamento tido como análogo ao de uma árvore de decisão.

Algoritmo 1: Pseudocódigo do *AdaBoost* .

Obtém N exemplos de imagens $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, onde x corresponde a matriz de pixels de uma imagem, e $y = 0$, ou 1 para exemplos negativos e positivos, respectivamente.

Inicializa os pesos $\omega_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_i = 0$ e 1 respectivamente, onde m é o total de exemplos negativos e l o total de positivos.

para t=1,...,T **faça**

  Normalize os pesos: $\omega_{t,i} \leftarrow \frac{\omega_{t,i}}{\sum_{j=1}^{N} \omega_{t,j}}$, onde ω_t é uma distribuição de probabilidade.

  para cada *característica* j **faça**

    Treine um classificador h_j restrito ao uso de uma única característica.

    Avalie o erro de acordo com: $\omega_j, \epsilon_j = \sum_i \omega_i |h_j(x_i) - y_i|$.

  Escolha o classificador h_t com o menor erro ϵ_t.

  Atualize os pesos: $\omega_{t+1,i} = \omega_{t,i} \beta_t^{(1-e_i)}$, onde $e_i = 0$, se o exemplo x_i for classificado corretamente, $e_i = 1$ caso contrário, e $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

O classificador forte é definido por: $H(x) = \begin{cases} 1, & \text{se } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0, & \text{caso contrário.} \end{cases}$,

onde $\alpha_t = \log \frac{1}{\beta_t}$.

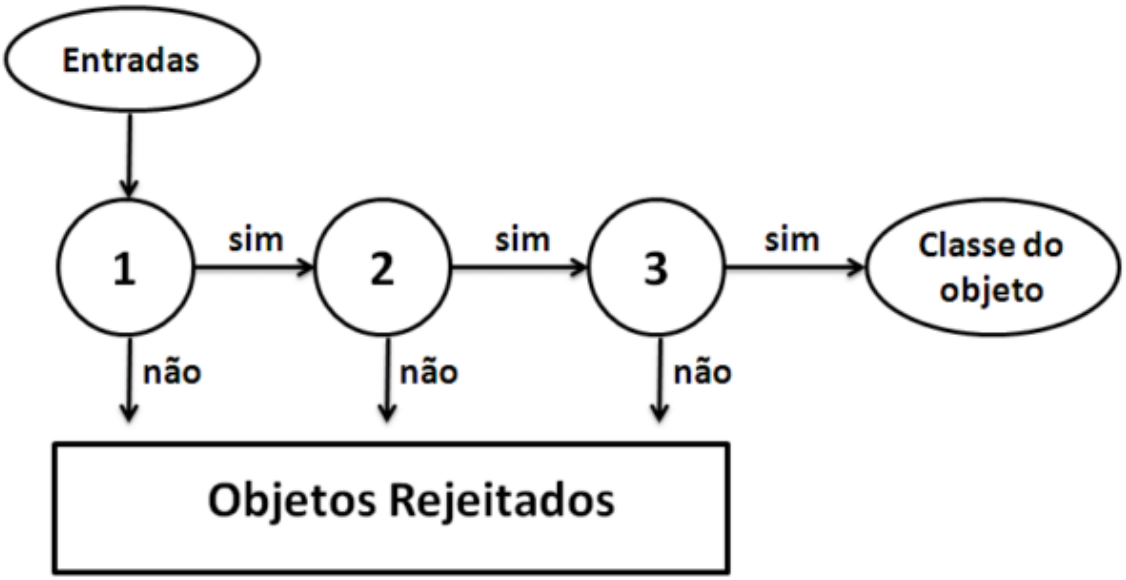

Figura 3 - Esquema de cascateamento de classificadores fracos para construção de de um classificador forte

Para realizar o treinamento da cascata de classificadores, duas taxas são computadas: a taxa de detecção D (ou de acerto) e a taxa de falso positivo F. A taxa de detecção depende do número de classificadores fracos K da cascata e da taxa de detecção de cada um dos classificadores fracos d_i. Pode-se ter, portanto, que a taxa de detecção é dada pela equação (2.7). Assim como a taxa de detecção, a taxa de falso positivo também deve considerar o número de classificadores fracos K, associada à taxa de falso positivo de cada classificador fraco, denotada por f_i, analogamente à taxa de detecção, como pode ser visto na equação (2.8),

$$D = \prod_{i}^{K} d_i \,, \tag{2.7}$$

$$F = \prod_{i}^{K} f_i \,. \tag{2.8}$$

O Algoritmo 2 descreve o pseudo-código do treinamento da cascata de classificadores.

Algoritmo 2: Pseudocódigo do treinamento de uma cascata de classificadores .

```
Seleciona os valores para: f: a máxima taxa de aceitação de falso positivo por
estágio da cascata. d: a taxa mínima de detecção por estágio aceita.
m = conjunto de exemplos negativos.
l = conjunto de exemplos positivos.
F_0 = 1,0; D_0 = 1,0; i = 0.
enquanto F_i > F_alvo faça
    i = i + 1.
    n_i = 0; F_i = F_{i-1}.
    enquanto F_i > f.F_{i-1} faça
        Usa l e m para treinar um classificador com n_i características usando o
        AdaBoost.
        Avalia a cascata de classificadores no conjunto de validação para
        determinar F_i e D_i.
        Diminui o limiar para o i-ésimo classificador até que a cascata atual tenha
        uma taxa de detecção de pelo menos d(D_{i-1}), o que também afeta F_i.
    m ← 0.
    se F_i > F_alvo então
        Avalia o detector da cascata atual no conjunto de imagens negativas e
        coloca qualquer detecção falsa no conjunto m.
```

2.2. Algoritmos Genéticos

Este algoritmo é inspirado na evolução dos genes, onde as características de cada indivíduo é representada utilizando genótipos. Proposto inicialmente por Fraser, foi popularizado devido aos trabalhos de John Holland [6]. O GA é composto por três operadores: de seleção, de cruzamento e de mutação.

Por ser um algoritmo robusto a mínimos locais devido ao operador de mutação que será descrito posteriormente, não ter necessidade de informação *a priori* sobre o problema, e robustez para lidar com descontinuidades na função de otimização, este algoritmo foi escolhido para ser aplicado ao problema.

Na versão canônica, proposta por Holland [6], a representação utilizava um vetor de bits, o critério de seleção era proporcional (não havia preferência entre os pais escolhidos), era utilizado um ponto para cruzamento e a mutação ocorria de maneira uniforme. As subseções seguintes abordam cada operador.

2.2.1. Operador de cruzamento

O cruzamento é o operador responsável pela combinação de informação de diferentes indivíduos da população. Esta combinação pode acontecer de três formas: de maneira sexuada, assexuada ou utilizando multi-recombinação.

Caso o cruzamento seja feito de modo assexuado, apenas um pai é utilizado para gerar um indivíduo candidato à geração seguinte. Um ou mais pais são escolhidos para dar origem aos novos indivíduos. Essa escolha é feita probabilisticamente e, geralmente, uma probabilidade alta taxa de cruzamento é utilizada. No caso em que *n* pais (com *n >1*) sejam utilizados para o cruzamento, deve-se evitar que o mesmo pai seja selecionado todas as *n* vezes para que não seja gerada uma cópia do indivíduo para prole.

No caso de utilização da representação binária dos cromossomos, além da quantidade de pais utilizada na criação de candidatos para a geração seguinte, há também três formas distintas para executar o cruzamento. Pode-se fazer uso de dois pontos para executar o cruzamento. Nesta abordagem, utiliza-se na região entre os dois pontos de corte informação proveniente de um dos pais e nas outras duas regiões utiliza-se os genes do outro pai, exemplificado na Figura 4(b). Outra forma de efetuar o cruzamento é fazê-lo de maneira uniforme, onde para cada item da representação do indivíduo (cada gene) é sorteado aleatoriamente gene de um dos pais, como é mostrado na Figura 4(c). É possível fazer uso do cruzamento com um ponto de corte, selecionado aleatoriamente, na representação dos genes, onde antes de um dado ponto são utilizados os genes de um dos pais e a partir do mesmo ponto é utilizada informação do outro pai, como é mostrado na Figura 4(d).

Já para o caso de representação decimal, é possível utilizar algumas abordagens, entre elas um operador linear proposto por Wright. Dados os pais $x_1(t)$ e $x_2(t)$, três indivíduos seriam gerados a partir das seguintes combinações: $[x_1(t)+x_2(t)]$, $[(1{,}5x_1(t)-0{,}5x_2(t))]$ e $[(-0{,}5x_1(t)+1{,}5x_2(t))]$. Após os indivíduos serem gerados, seus *fitness* são avaliados e os dois melhores indivíduos tornam-se candidatos à geração seguinte.

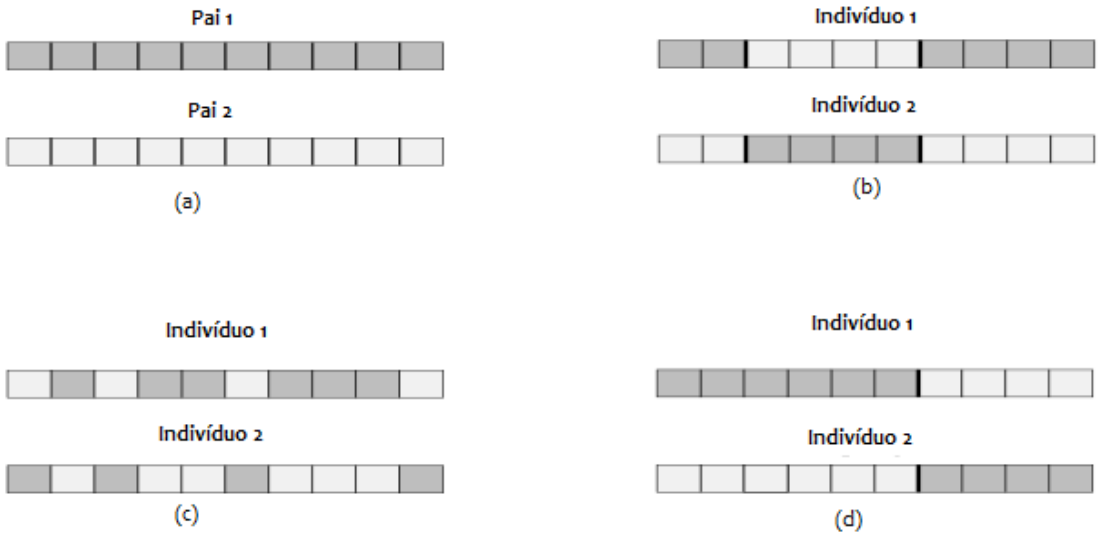

Figura 4 - Exemplo de cruzamento: (a) pais utilizados, (b) cruzamento com dois pontos de corte, (c) cruzamento uniforme e (d) cruzamento com um ponto de corte.

2.2.2. Operador de mutação

A função principal da mutação é adicionar novo material genético aos indivíduos, adicionando diversidade à população. Pode-se aplicar probabilidades distintas para cada cromossomo de um indivíduo ou utilizar a mesma para todos os cromossomos. No caso de utilizar a mesma probabilidade (p_i), a probabilidade do indivíduo sofrer mutação é dada por

$$prob(x(t)) = 1-(1-p_i)^n, \qquad (2.9)$$

onde *n* é o número de genes de um indivíduo.

Para representações binárias, a mutação pode ocorrer de três formas: uniforme (aleatória), mutação *inorder* e mutação gaussiana. Na mutação uniforme, posições aleatórias são escolhidas para ter o valor de seu gene alterado por um valor aleatório. Na mutação *inorder* são escolhidos aleatoriamente dois pontos, em algo semelhante ao cruzamento com dois pontos de corte, e entre esses pontos estão os genes que sofrerão mutação.

Por último, se tem a representação gaussiana, proposta por Hinterding [14], onde valores entre [0,1] são obtidos através da distribuição de Poisson para decidir quais genes sofrerão mutação. A Figura 5(a) exemplifica um caso de mutação uniforme, enquanto que a Figura 5(b) apresenta um caso de mutação com dois pontos de corte.

Em problemas de alta dimensionalidade, a mutação pode aumentar significativamente o custo computacional do GA. Numa tentativa de diminuir esse custo, Birru dividiu os genes e associou cada parte do cromossomo a um valor e para executar a mutação em cada uma dessas partes.

Para representação fazendo uso de ponto flutuante, uma das primeiras propostas foi utilizar o seguinte operador baseado em um valor aleatório α (que pode ser 0 ou 1) gerado através de:

$$x'_{ij}(t) = \tilde{x}(t) + \Delta(t, x_{\max} - \tilde{x}_{ij}(t)) \text{, caso } \alpha = 0, \tag{2.10}$$

$$x'_{ij}(t) = \tilde{x}(t) + \Delta(t, \tilde{x}_{ij}(t) - x_{\min}) \text{, caso } \alpha = 1. \tag{2.11}$$

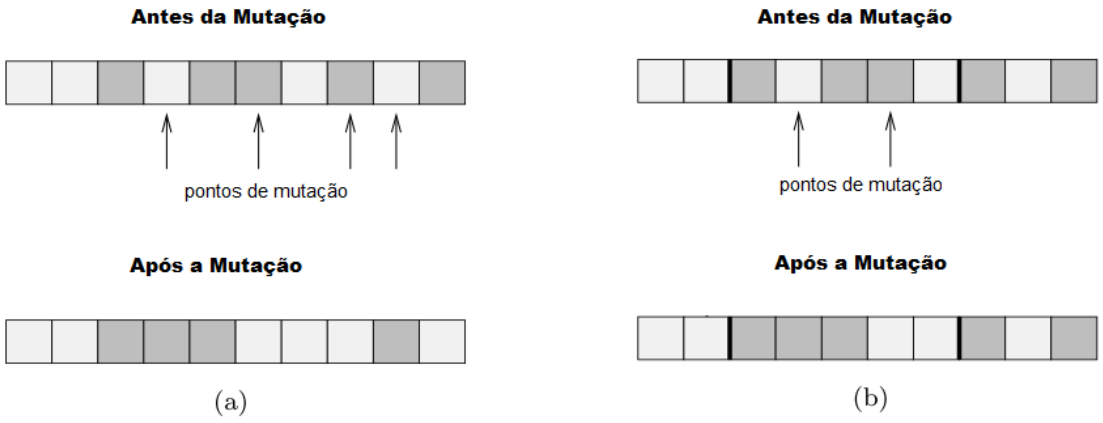

Figura 5 - Exemplo de mutação binária: (a) acontecendo por mutação aleatória, e em (b) mutação do tipo *inorder.*

2.2.3. Operador de seleção

Para efetuar a seleção, podem ser seguidas algumas abordagens. Uma abordagem comum de se seguir é utilizando elitismo, onde os indivíduos com melhor avaliação na função objetivo são selecionados entre a população e os novos indivíduos criados para fazer parte da geração na próxima iteração. Fazendo uso dessa abordagem, o algoritmo tende a convergir de maneira mais rápida, porém perde em diversidade e pode ficar preso em mínimos locais mais facilmente.

Outra forma de efetuar a seleção é compondo com indivíduos sem observar a avaliação da função objetivo. Utilizar este tipo de operação gera maior diversidade, e torna mais improvável que o algoritmo fique preso em mínimos locais. Por outro lado, a convergência pode ser mais lenta. Além disso, pode ser que o melhor indivíduo em uma dada iteração não seja selecionado para a próxima, causando perda de informação relevante.

O Algoritmo 3 descreve o pseudocódigo do algoritmo proposto por Holland [6].

Algoritmo 3: Pseudocódigo do *Genetic Algorithm* .

```
1  Inicializa os individuos com valores aleatórios;
2  while (condiçao de parada não for atingida) do
3      for (cada indivíduo) do
4          avalie a função objetivo;
5          selecione os pais;
6          gere novos individuos através de crossover e mutação;
7          avalie o fitness dos novos indivíduos gerados;
8          forme a populaçao para o proximo ciclo seguindo o critério de seleção;
9      end
10 end
```

2.3. Busca por enxames de partículas

Desenvolvido por Kennedy e Ebehart [16], o PSO (*Particle Swarm Optimization*) é um algoritmo baseado em enxames de partículas que pode ser implementado inicialmente podendo fazer uso de um dos dois tipos de topologia: global e local. Na topologia global está presente a noção de g_{Best} (*global best*) baseada na topologia em estrela, mostrada na Figura 6 (a), onde todos os nós se ligam a todos os outros. A topologia local, por sua vez, apresenta a noção de l_{best} (*local best*), baseada na topologia em anel, exemplificada na Figura 6(b). A diferença causada por essa escolha de topologia será descrita nas Seções 2.3.1 e 2.3.2, respectivamente [16].

Assim como no GA, um enxame (por se tratar de partícula, dá-se o nome de enxame ao equivalente à população) é mantido e cada partícula é uma possível solução. Da mesma forma que no GA, o PSO é inicializado com valores aleatórios, porém com um vetor de velocidades associado à cada partícula para cada dimensão do problema [6].

Juntamente com o conceito de velocidade, tem-se a noção de que as partículas se deslocam pelo espaço de busca. A posição da partícula, denotada por $\vec{x_i}(t)$ deve ser entendida como a composição da posição anterior e a velocidade atual $\vec{v}_i(t)$ da partícula, como mostra a equação (2.12), que considera o intervalo de tempo unitário. O cálculo da velocidade, por sua vez, depende da topologia escolhida para a implementação do PSO e do p_{best} de cada partícula. O p_{best} é a posição com melhor avaliação de função de aptidão por onde a partícula já esteve.

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1). \tag{2.12}$$

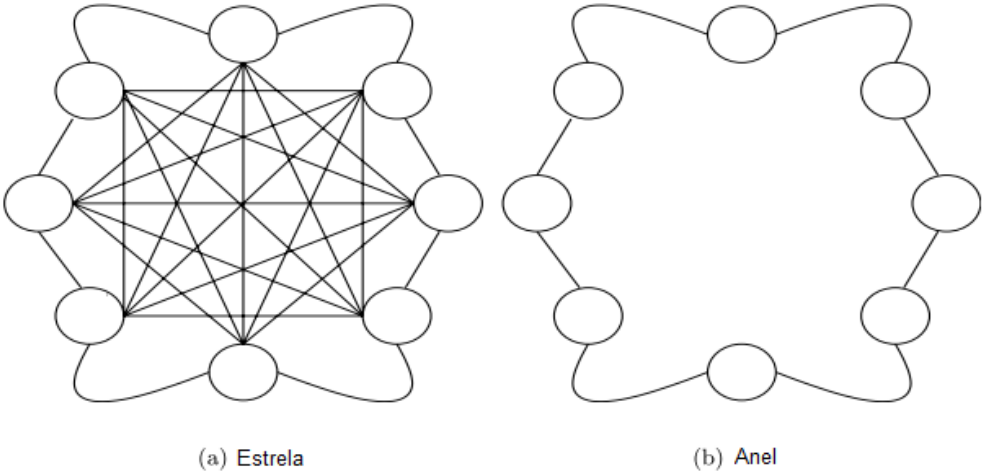

Figura 6 - Topologias do PSO: (a) topologia em estrela e (b) topologia em anel.

2.3.1. g_{best} PSO

Na topologia global todas as partículas têm acesso à informação sobre a melhor posição já encontrada pelo enxame como um todo. Para o g_{best}, a velocidade é calculada seguindo a equação (2.13), descrita abaixo.

$$\vec{v}_i(t+1) = \chi[\vec{v}_i(t) + c_1 r_1(\vec{x}_i(t) - \vec{p}_i(t)) + c_2 r_2(\vec{x}_i(t) - \vec{l}_i(t))] \quad (2.13)$$

,

onde χ é o fator de constrição utilizado para ajustar a influência das velocidades anteriores das partículas no processo de busca e pode ser calculado utilizando a equação

$$\chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}}, \varphi = c_1 + c_2 \quad (2.14)$$

,

enquanto que $\vec{l}_i(t)$, por sua vez, representa a melhor posição entre os vizinhos da partícula *i*. Por se tratar do g_{best} PSO, os vizinhos são todas as partículas do enxame. $\vec{v}_i(t)$ denota, por sua vez a velocidade da partícula i na dimensão j, $\vec{x}_i(t)$ é a posição da partícula i na dimensão j no instante *t*, c_1 e c_2 são constantes usadas pra dimensionar a contribuição cognitiva e social e

$r_{1j}(t)$ e $r_{2j}(t)$ são valores aleatórios pertencentes ao intervalo [0,1], responsáveis por introduzir um componente estocástico ao algoritmo. O componente $\vec{p}_i(t)$ denota a melhor posição alcançada pela partícula i (seu p_{best}) desde o início do algoritmo.

Tendo sido explicados os parâmetros da equação (2.13), o componente cognitivo da partícula é dado por $c_1 r_1[\vec{x}_i(t) - \vec{p}_i(t)]$, enquanto que o componente social é dado por $c_2 r_2[\vec{x}_i(t) - \vec{l}_i(t)]$.

O componente cognitivo é responsável por avaliar o desempenho da partícula no momento atual em relação ao seu desempenho em iterações anteriores. Como efeito, a partícula é levada de encontro à sua melhor posição durante todas as iterações. O componente social, por sua vez, avalia a partícula em relação à melhor partícula do enxame, fazendo com que ela tenha uma componente na direção da melhor posição já encontrada pela sua vizinhança. Há ainda o componente inercial da partícula, denotado por $\vec{v}_i(t)$, que tende a manter a partícula deslocando-se na mesma direção do instante de tempo anterior.

A melhor posição global em problemas de otimização quando se pretende minimizar o valor da função objetivo é dada pela equação (2.15), devendo-se considerar todas as melhores posições encontradas por cada partícula desde a primeira iteração. Analogamente, em problemas de maximização, onde se procura maximizar os valores, só é necessário buscar pelo máximo entre os melhores locais de todas as partículas desde a primeira iteração do algoritmo buscando maximizar o valor da função objetivo.

$$\hat{y}(t) = \min\{ f(y_i(t)\}. \tag{2.15}$$

2.3.2. l_{best} PSO

Como foi mencionado anteriormente, o l_{best} baseia-se na topologia em anel. Neste tipo de implementação, as partículas só tem informações sobre seus

vizinhos mais próximos. Isso quer dizer que a *i*-ésima partícula só tem acesso ao p_{best} das partículas $i+1$ e $i-1$. A equação para o cálculo da velocidade é idêntica à equação (2.13), porém com o termo $\overset{*}{l}_i(t)$ sendo apenas a posição que retorne o melhor *fitness* entre o *pbest* da própria partícula e do seu anterior e sucessor imediatos.

Nesta abordagem, o componente social da velocidade direciona a partícula em busca da melhor posição já descoberta por ela e seus vizinhos. O Algoritmo 4 abaixo descreve a implementação l_{best} *PSO*. O pseudocódigo do l_{best} está apresentado porque este foi um dos algoritmos utilizados para o treinamento do *AdaBoost*.

Algoritmo 4: Pseudocódigo do PSO.

```
1  Inicializa as partículas com valores aleatórios;
2  enquanto (condiçao de parada não for atingida) faça
3      para (cada partícula) faça
4          avalie a partícula usando a função objetivo;
5          caso o novo fitness seja melhor do que seu pbest, atualize o pbest;
6      para (cada partícula) faça
7          atualize o valor de lbest;
8          atualize a velocidade da partícula;
9          atualize a posição da particula;
10 fim.
```

2.3.3. Implicações na escolha entre g_{best} PSO e l_{best} PSO

Devido ao tipo de topologia de cada um dos dois tipos de PSO, o global tem uma maior interconectividade entre as partículas, levando a uma convergência mais rápida, porém ao custo de ter uma menor diversidade. Isso torna o g_{best} PSO mais suceptível a ficar preso em mínimos locais, fazendo com que o l_{best} PSO tenha, de maneira geral, melhores soluções em termos de qualidade em problemas multimodais.

2.4. Algoritmo de busca por cardumes

O algoritmo de Busca por Cardumes (FSS, *Fish School Search*) é um algoritmo de enxames criado por Bastos-Filho e outros [7] inspirado no

comportamento de cardumes de peixes. Duas situações fazem os peixes agirem coletivamente: buscando proteção contra predadores e buscando realizar tarefas coletivamente, como buscar alimento. Posto isso, três comportamentos presentes nos peixes formam o núcleo deste algoritmo: alimentação, nado individual e nado coletivo que serão explicados em subseções mais adiante.

Como principal diferença entre o FSS e o PSO é a forma de representação de sucesso. Ao invés de ter a partícula salvando sua melhor posição, o peixe mais bem sucedido é o peixe mais pesado em uma dada iteração do algoritmo.

O FSS tem como princípio básico gerar um baixo custo computacional devido às operações realizadas em cada indivíduo. Além disso, a comunicação e o controle entre os peixes é descentralizado e o aprendizado adaptativo.

Outro conceito interessante é o de alimento. Este conceito está relacionado com a função objetivo. Em problemas onde se quer buscar maximização, a quantidade de alimento é diretamente proporcional à avaliação da função objetivo. Analogamente, em problemas de minimização, regiões com mínimos locais contêm mais alimento do que as demais.

2.4.3. Operador de alimentação

Este operador é responsável por fazer o controle do peso do peixe W_i baseado na função objetivo f. Esse ganho foi proposto ser proporcional à normalização da diferença entre a avaliação da função objetivo na atual posição, denotada por $f[x_i(t+1)]$, e da avaliação da função objetivo no instante (iteração) anterior $f[x_i(t)]$ para o i-ésimo peixe do cardume. Sendo assim, em caso de melhora o peixe acaba por engordar, e em caso de piora ele acaba por perder peso. Essa relação pode ser vista na equação

$$W_i(t+1) = W_i(t) + \frac{f[x_i(t+1)] - f[x_i(t)]}{\max\{|\ f[x_i(t+1)] - f[x_i(t)]\ |\}} \tag{2.16}$$

.

Para assegurar que o algoritmo convirja, algumas medidas devem ser tomadas. A criação de um parâmetro adicional para limitar o peso máximo do peixe (W_{scale}) para o peso permitido do peixe variar apenas entre “1” e W_{scale}. Além disso na inicialização do algoritmo todos os peixes devem ter seu peso inicializados com $\frac{W_{scale}}{2}$ e a avaliação da função de aptidão de cada peixe deve ocorrer apenas uma vez a cada iteração.

2.4.4. Operadores de natação

Esses operadores são responsáveis pela movimentação dos peixes do cardume. O movimento pode se dar de três formas: devido ao seu movimento individual, devido ao movimento coletivo instintivo ou ao movimento coletivo volitivo.

No movimento de nado individual, a direção do movimento é escolhida aleatoriamente. Dado que o ponto de destino esteja dentro dos limites do espaço de busca, caso o ponto de destino tenha uma quantidade de alimento (representando uma melhor avaliação da função objetivo), o peixe desloca-se para esta nova posição. Caso o local não proveja um ambiente com maior quantidade de alimento, o peixe permanece na sua posição atual.

Deve-se determinar como parâmetro o tamanho do passo individual máximo a ser dado pelos peixes $step_{ind}$. O tamanho do passo individual pode ser decrementado com o passar das iterações para promover uma busca mais refinada (em profundidade). Para se ter uma maior aleatoriedade, o valor do passo deve ser multiplicado por um valor aleatório pertencente ao intervalo [0,1] gerado através de uma distribuição uniforme. Esse tipo de movimento deve ocorrer para cada peixe do cardume em todos os ciclos do algoritmo. Após todos os peixes fazerem seus movimentos individuais, os mesmos devem ser alimentados.

O movimento coletivo instintivo deve ocorrer em seguida e é baseado no sucesso dos peixes de todo o cardume. Isto significa que o sucesso (ou fracasso) dos peixes como um todo no movimento individual influenciam nesta etapa da sua movimentação. Deve-se determinar a direção para onde o cardume, como um todo, deve ser deslocado e então os peixes devem ser deslocados conforme a equação (2.17).

$$\overset{\rho}{x}_i(t+1) = \overset{\rho}{x}_i(t) + \frac{\sum_{i=1}^{N} \Delta \overset{\rho}{x}_{ind} \{ f[x_i(t+1)] - f[x_i(t)] \}}{\sum_{i=1}^{N} \{ f[x_i(t+1)] - f[x_i(t)] \}} \quad (2.17)$$

,

em que $\Delta \overset{\rho}{x}_{ind}$ é a variação da posição do peixe i devido ao movimento individual.

O movimento volitivo coletivo, por sua vez, deve ocorrer após o coletivo instintivo. Esse movimento ocorre de acordo com a média do peso do cardume. Em caso de aumento do peso do cardume como um todo, o seu raio deve contrair para uma busca em profundidade. Caso contrário, o raio do cardume deve dilatar para explorar outras áreas, fazendo uma busca em largura.

Cada peixe deve sofrer uma pequena variação na sua posição levando em conta o baricentro do cardume, obtido a partir da posição dos peixes e seu peso atual, conforme mostra a equação (2.18).

$$Bari(t) = \frac{\sum_{i=1}^{N} \overset{\rho}{x}_i(t)\, W_i(t)}{\sum_{i=1}^{N} \overset{\rho}{x}_i(t)} . \quad (2.18)$$

Outro parâmetro, o passo volitivo $step_{voli}$, é necessário para promover a movimentação do cardume. Caso o peso do cardume tenha aumentado, o

cardume deve contrarir segundo a equação (2.19). Caso contrário, o cardume deve se expandir seguindo a equação (2.20).

$$\overset{\rho}{x}_i(t+1) = \overset{\rho}{x}_i(t) - step_{voli} \cdot rand[\overset{\rho}{x}_i(t) - Bari(t)] \quad (2.19)$$

,

$$\overset{\rho}{x}_i(t+1) = \overset{\rho}{x}_i(t) + step_{voli} \cdot rand[\overset{\rho}{x}_i(t) - Bari(t)] \quad (2.20)$$

.

Novamente, pode-se utilizar o tamanho do passo sendo decrementado ao longo das iterações. Assim como no movimento individual, o operador possui um termo aleatório (*rand*), que também é gerado por uma função densidade de probabilidade uniforme no intervalo [0,1].

O algoritmo 5 apresenta o pseudocódigo do FSS.

Algoritmo 5: Pseudocódigo do FSS.

```
Inicializa o cardume com valores aleatórios;
enquanto (condiçao de parada não for atingida) faça
    para (cada peixe) faça
        avalie a função objetivo;
        execute o movimento individual;
    para (cada peixe) faça
        execute o operador de alimentação;
    para (cada peixe) faça
        execute o movimento coletivo instintivo;
    calcule o baricentro do cardume;
    para (cada peixe) faça
        execute o movimento coletivo volitivo;
        avalie a função objetivo;
    atualize o passo individual;
fim.
```

3. Metodologia utilizada

Neste capítulo serão descrito como foram separados os conjuntos de treinamento da base de dados, como foram desenvolvidos os sistemas de detecção para o treinamento *AdaBoost* e, por fim, como foi desenvolvida a etapa de teste.

Os sistemas foram desenvolvidos utilizando a linguagem de programação C/C++ e o ambiente de desenvolvimento escolhido foi o eclipse, com o compilador MinGW. As simulações foram realizadas em um computador com 4GB de memória RAM, processador Intel Core 2 Quad 2,66 GHz e utilizando o sistema operacional Windows 7.

O banco de imagens foi dividido em dois subgrupos: um contendo 75% das imagens que foram utilizadas para fazer o treinamento, enquanto que os 25% restantes das imagens foram utilizadas como banco de testes. Para ambos os grupos foram utilizadas imagens positivas, ou seja, que contêm plaquetas, e imagens negativas, que não contêm plaquetas. Na etapa de treinamento, os 75% das imagens positivas correspondem a 110 imagens de um total de 147. Enquanto que as negativas correspondem a 417 imagens de um total de 556. A base de testes contém as imagens restantes.

Para efeito comparativo, foram utilizadas as três técnicas para treinar o *AdaBoost* com 5, 10 e 20 classificadores fracos para compor um classificador forte. Para cada treinamento foram utilizadas 30 partículas (PSO), 30 indivíduos (GA) e 30 peixes (FSS). Quatro tipos de características foram utilizadas (descritas na Figura 1). Cada simulação fez uso de dez mil iterações e foram executadas trinta simulações para cada experimento. Como função objetivo em todos os treinamentos, foi utilizada a diferença entre a taxa de acerto e a taxa de falso positivo.

Nas próximas sessões serão descritas as particularidades de cada algoritmo para o treinamento do *AdaBoost*.

3.1. Detecção de plaquetas utilizando o GA

Para o treinamento do *AdaBoost* utilizando GA foram utilizados 30 indivíduos para compor a população e foram criados 15 indivíduos na fase de cruzamento e mutação. Durante o cruzamento, os pares de pais foram escolhidos aleatoriamente, tomando cuidado para que os pais i e j de um novo indivíduo k não fossem o mesmo indivíduo ($i \neq j$). O cruzamento foi realizado segundo a equação (3.1)

$$classifica\ \ dor_k[n] = \begin{cases} classifica\ \ dor_i[n], & se\ rand < 0{,}5 \\ classifica\ \ dor_j[n], & caso\ contrári\ i\ o \end{cases} \quad (3.1)$$

.

A taxa de mutação m_{tax} foi inicializada em 5% e era decrementada de 1% a cada mil iterações para prover diversidade no início do algoritmo e evitar perda de conhecimento nas iterações finais. Além disso, quando ocorria a mutação em um novo indivíduo apenas um de seus classificadores fracos era alterado por completo. Uma nova característica era atribuída a ele, assim como nova posição e tamanho da característica do classificador.

Na seleção, por sua vez, foi utilizada a escolha aleatória de indivíduos, porém ao invés de selecionar trinta indivíduos para compor a nova população, foram escolhidos apenas vinte e sete. Os três indivíduos restantes são os dois melhores da população atual e o melhor indivíduo gerado através do processo de reprodução.

O Algoritmo 6 descreve o pseudocódigo utilizado para o treinamento do *AdaBoost*.

Algoritmo 6: Pseudocódigo do treinamento *AdaBoost* utilizando o GA.

```
1  Carrega a base de treino;
2  Inicializa os individuos com valores aleatórios para os classificadores;
3  para (cada indivíduo) faça
4      calcule o limiar do AdaBoost;
5      calcule a polaridade do classificador;
6  enquanto (condiçao de parada não for atingida) faça
7      para (cada indivíduo) faça
8          avalie a função objetivo do AdaBoost;
9          gere novos individuos através de crossover e mutação;
10         avalie o fitness dos novos indivíduos gerados;
11         forme a populaçao para o proximo ciclo seguindo o critério de seleção explicado
           nesta seção;
12         salva o melhor individuo da populaçao como melhor individuo já encontrado;
13 fim.
```

3.2. Detecção de plaquetas utilizando o PSO

Para o treinamento utilizando o l_{best} PSO, foram utilizadas 30 partículas. A velocidade dos componentes dos classificadores foram limitadas. Para a variação de tipo de características esse limite foi de 3 (são quatro características). Para os demais parâmtros (posição no eixo *X*, posição no eixo *Y*, largura e altura) as velocidades foram limitadas em [-10,10], tendo em vista que cada um desses parâmetros podem atingir valores de entre [0,23]. Além desses parâmetros, os valores das constantes c_1 e c_2 foi 2,05 e o valor do fator de constrição χ utilizado foi 0,72984. O Algoritmo 7 descreve o pseudocódigo utilizado.

Algoritmo 7: Pseudocódigo do treinamento *AdaBoost* utilizando o PSO.

```
carrega a base de treino;
Inicializa as particulas com valores aleatórios para os classificadores;
para (cada particula) faça
    calcule o limiar do AdaBoost;
    calcule a polaridade do classificador;
enquanto (condiçao de parada não for atingida) faça
    para (cada particula) faça
        avalie a função objetivo do AdaBoost;
        if (f(x) > pbest) then
            atualiza pbest;
            if (f(x) > pbest da melhor partícula) then
                salva a partícula como a melhor partícula ate então;
    para (cada particula) faça
        encontre o lbest;
        atualize a velocidade da partícula;
        atualize a posição da partícula;
fim.
```

3.3. Detecção de plaquetas utilizando o FSS

Na detecção utilizando o FSS foram definidos os seguintes parâmetros: $W_{scale} = 20$ (peso máximo dos peixes), o que implica que todos os peixes são inicializados com $W = 10$. O peso mínimo dos peixes definido foi '1'. Para o tamanho máximo do passo individual de cada peixe foram atribuídos os seguintes valores: '1' para o tipo de característica de cada classificador, '2' para os demais parâmetros (posição no eixo *X*, posição no eixo *Y*, altura e largura).

O tamanho máximo do passo coletivo volitivo teve também '1' atribuído ao parâmetro relativo ao tipo de característica de cada classificador e aos demais parâmetros (posição no eixo *X*, posição no eixo *Y*, altura e largura) foram atribuídos o valor '3'. O Algoritmo 8 traz o pseudocódigo utilizado para fazer o treinamento *AdaBoost* utilizando FSS.

Algoritmo 8: Pseudocódigo do treinamento *AdaBoost* utilizando o FSS.

```
1  carrega a base de treino;
2  Inicializa as particulas com valores aleatórios para os classificadores;
3  para (cada particula) faça
4      calcule o limiar do AdaBoost;
5      calcule a polaridade do classificador;
6  enquanto (condiçao de parada não for atingida) faça
7      para (cada peixe) faça
8          avalie a função objetivo do AdaBoost;
9          execute o movimento individual;
10     para (cada peixe) faça
11         execute o operador de alimentação;
12     para (cada peixe) faça
13         execute o movimento coletivo instintivo;
14     calcule o baricentro do cardume;
15     para (cada peixe) faça
16         execute o movimento coletivo volitivo;
17         avalie a função objetivo;
18     atualize o passo individual;
19 fim.
```

4. Resultados Obtidos

4.1. Resultados utilizando 5 classificadores fracos

A Figura 7 mostra o gráfico da convergência dos algoritmos ao longo das iterações. Pode-se ver que a avaliação da função objetivo, representada por *vp - fp* (onde *vp* é a taxa de acerto ou verdadeiro positivo e *fp* é a taxa de falso positivo) as três técnicas apresentaram resultados semelhantes sobre qual delas seria a melhor a ser utilizada ao fim das dez mil iterações. Por outro lado, pode-se notar que a curva descrita para o GA aparenta ainda estar em crescimento, enquanto que os outros dois algoritmos desenvolvem um crescimento mais lento, podendo ter estagnado em algum máximo local.

Analisando o gráfico da Figura 8, que representa a dispersão entre as taxas de acerto encontradas por cada algoritmo, temos que o FSS obteve uma taxa de acerto média maior, enquanto que a taxa de acerto atingida pelo GA foi maior do que a do PSO. Vale observar também que a dispersão entre os resultados encontrados pelo GA foi menor.

Fazendo a análise do gráfico da Figura 9, pode-se observar a taxa de falso positivo obtida por cada algoritmo no treinamento *AdaBoost*. A menor taxa encontrada na média foi a obtida através do treinamento do GA, que também apresentou menor dispersão entre seus resultados. Como segunda menor taxa de falso positivo e de dispersão está o PSO, deixando o FSS como pior algoritmo para essa métrica.

Ao analisar o conjunto dos três gráficos obtidos, pode-se ter o GA como melhor algoritmo dentre os testados para cinco classificadores, apresentando-se como solução mais robusta.

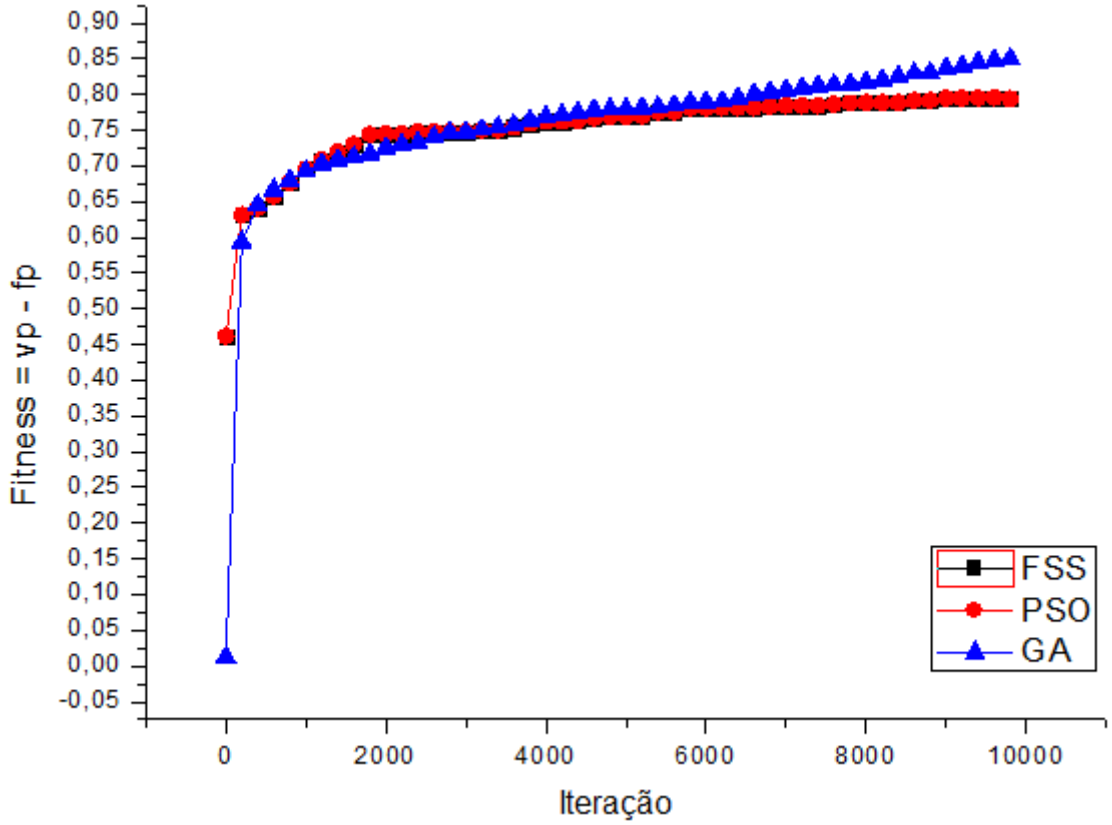

Figura 7 - Comparativo entre o *fitness* das técnicas implementadas ao longo das iterações para 5 classificadores.

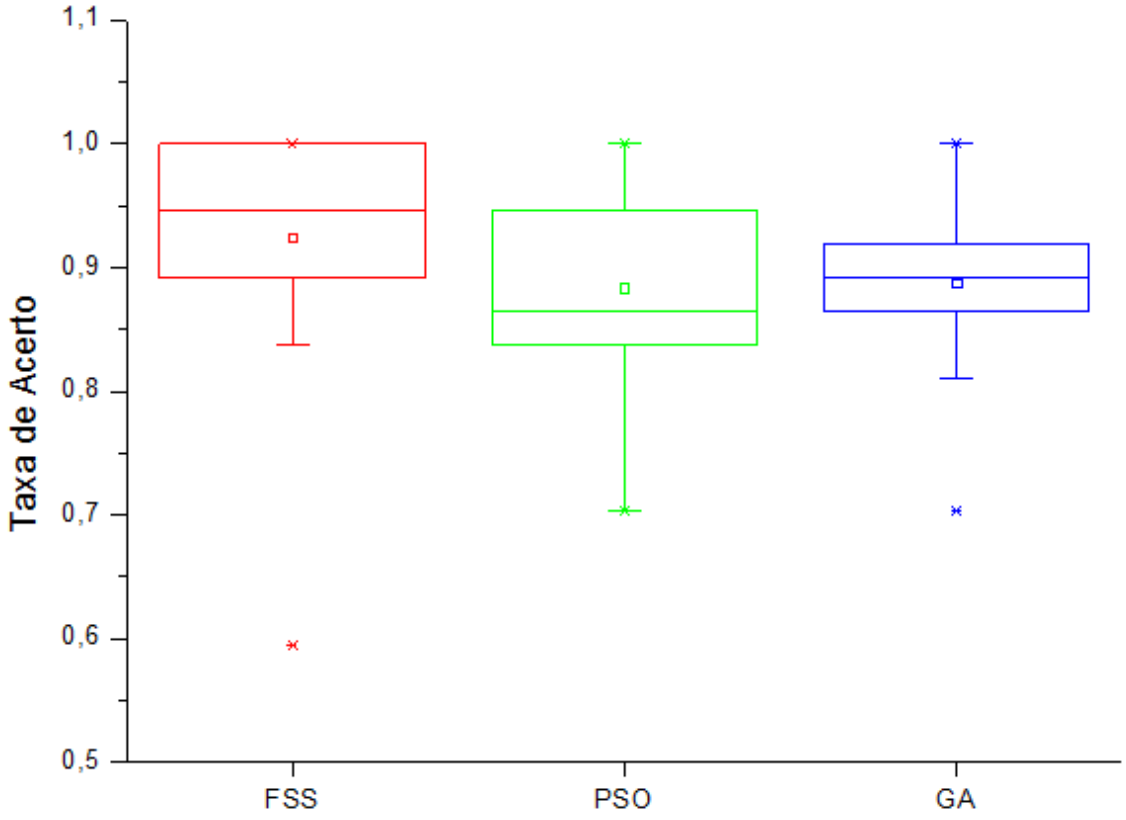

Figura 8 - Comparativo entre as taxa de acerto utilizando 5 classificadores fracos.

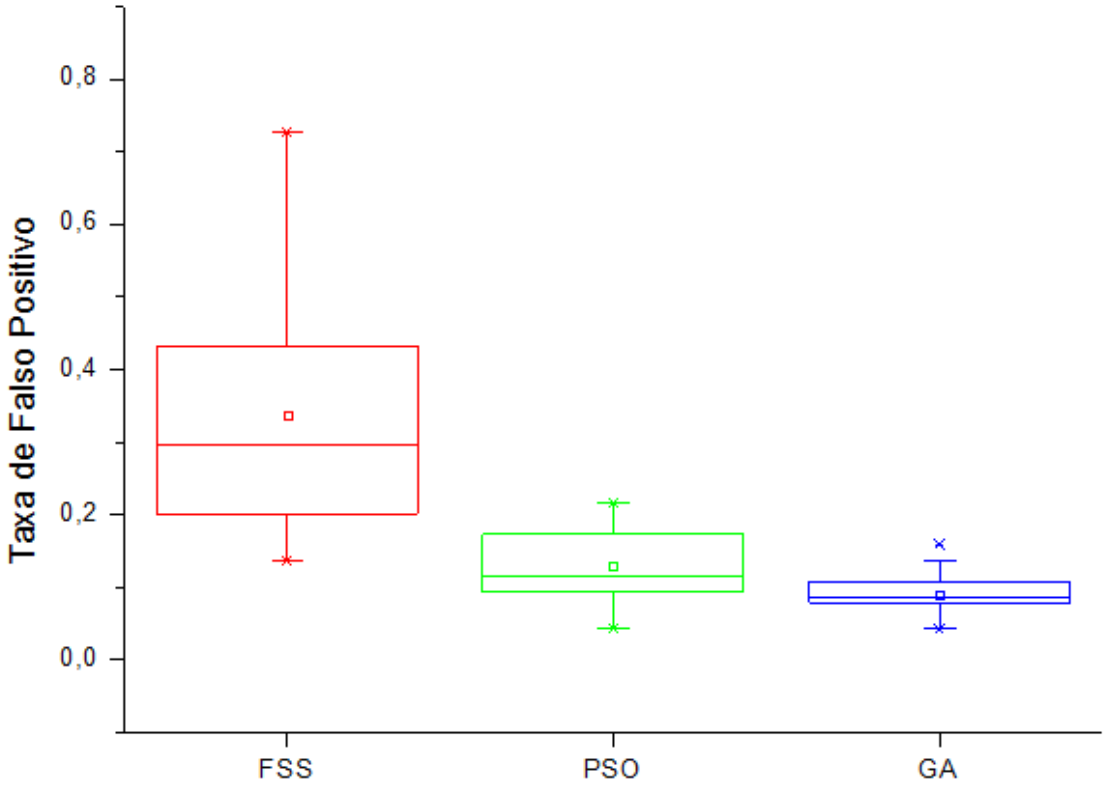

Figura 9 - Comparativo entre as taxa de falso positivo utilizando 5 classificadores fracos.

4.2. Resultados utilizando 10 classificadores fracos

Fazendo a mesma análise da Seção 4.1, temos que pelo gráfico de convergência (Figura 10) o GA e o PSO resultados semelhantes novamente e melhores do que os alcançados pelo FSS. O PSO aparentemente já havia estagnado, enquanto que o GA demonstra uma tendência de crescimento, tal qual mostrado na Seção 4.1.

Analisando o gráfico da Figura 11, que faz um comparativo entre as taxas de acerto encontradas por cada algoritmo, pode-se perceber que, novamente, o FSS obteve a melhor média e, dessa vez, também obteve os resultados mais uniformemente distribuídos. O resultado do GA mostra que o treinamento do *AdaBoost* obteve uma média melhor do que a do PSO e com os resultados menos dispersos.

Por fim, analisando o gráfico da Figura 12 pode-se observar que tal qual no treinamento para cinco classificadores fracos, o GA obteve uma menor taxa de falso positivo, o que o levou a ter a melhor avaliação no gráfico de convergência e demonstrar que sua escolha foi a mais adequada para o problema. O PSO teve a segunda menor taxa de falso positivo, enquanto que a taxa de falso positivo do FSS foi quase o dobro da média do GA, tornando o algoritmo com a pior avaliação na função objetivo.

Fazendo agora uma análise conjunta dos três gráficos, pode-se concluir que, novamente, o GA é a solução mais adequada para o caso de 10 classificadores, deixando o PSO como uma segunda alternativa.

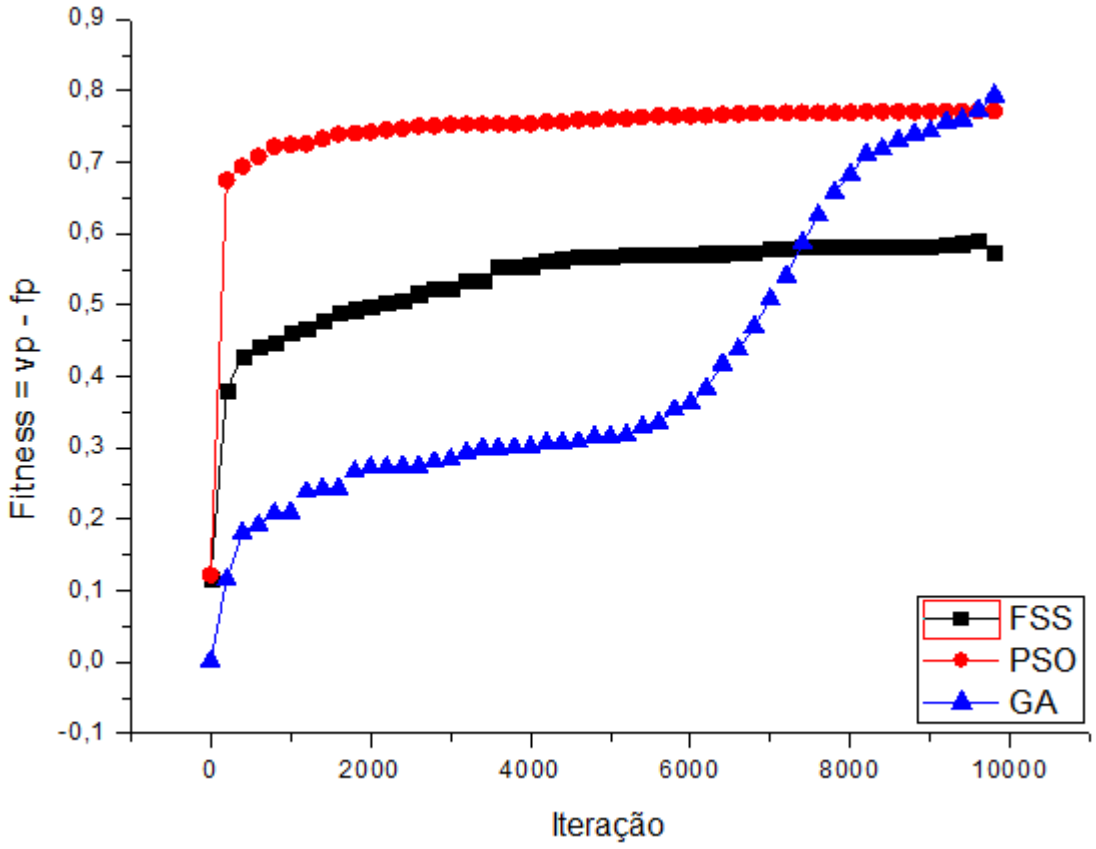

Figura 10 - Comparativo entre o *fitness* das técnicas implementadas ao longo das iterações para 10 classificadores fracos.

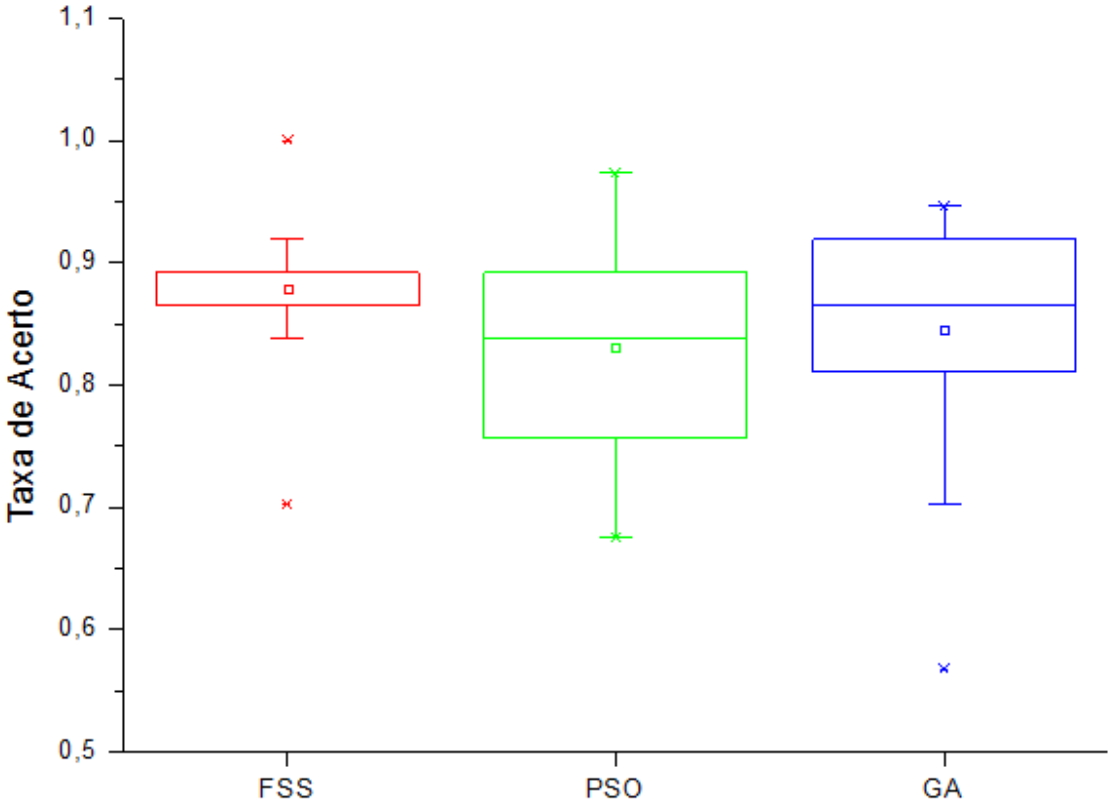

Figura 11 - Comparativo entre as taxas de acerto utilizando 10 classificadores fracos

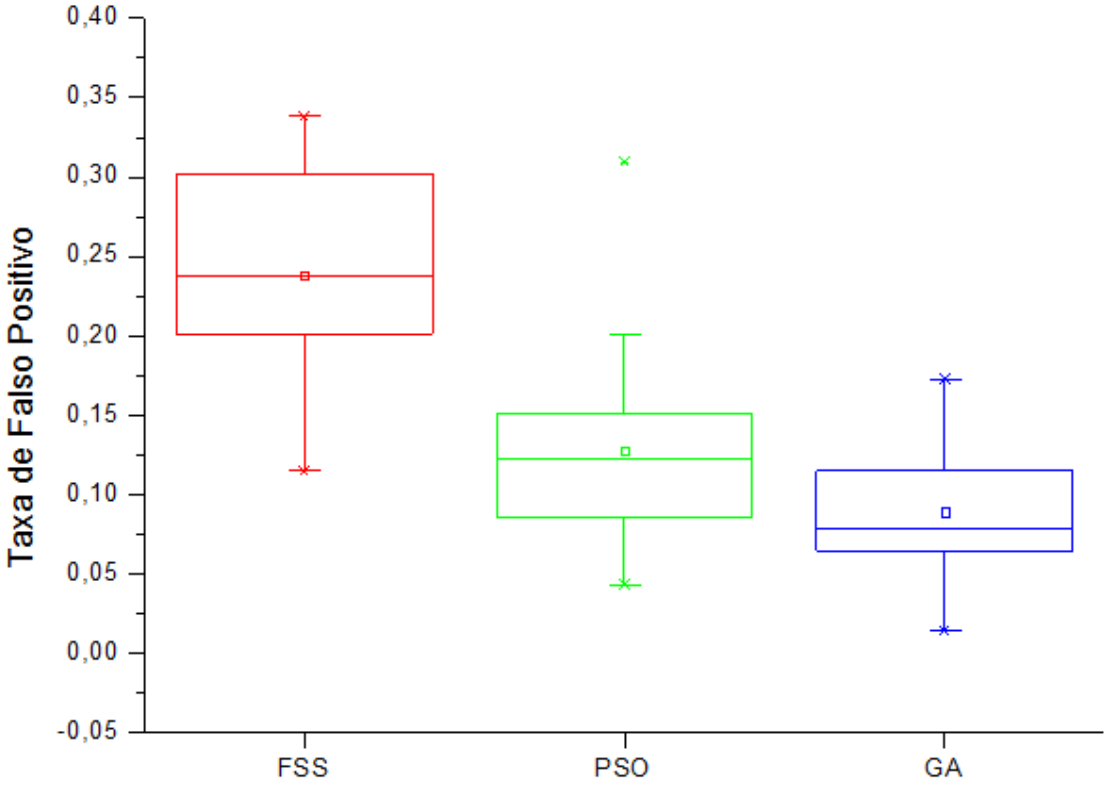

Figura 12 - Comparativo entre as taxas de falso positivo utilizando 10 classificadores fracos.

4.3. Resultados utilizando 20 classificadores fracos

A Figura 13 mostra o gráfico da convergência dos algoritmos ao longo das iterações. Pode-se ver que a avaliação da função objetivo obteve um maior valor para o treinamento desempenhado pelo PSO, enquanto que o FSS obteve o segundo melhor resultado e o GA o pior resultado encontrado para as três quantidades de classificadores fracos. Possivelmente o GA não conseguiu lidar com a alta dimensionalidade do problema, precisando de mais iterações que as demais técnicas testadas para apresentar evolução.

Vale ressaltar que foram usados os mesmos parâmetros para todas as quantidades de classificadores para as três quantidades de classificadores. É possível que apenas alterando os parâmetros seja possível encontrar melhores resultados com todas as três técnicas.

Outro ponto importante é que, assim como nos casos anteriores (apresentados nas Seções 4.1 e 4.2), ao fim das dez mil iterações o GA apresenta uma tendência de crescimento, que não se pode observar para os outros dois algoritmos. Talvez se mais iterações fossem utilizadas o GA atingisse melhores resultados do que as outras técnicas.

Analisando o gráfico da Figura 14, que representa a dispersão entre as taxas de acerto encontradas por cada algoritmo, temos que o PSO obteve a maior taxa de acerto média, enquanto que a taxa de acerto atingida pelo GA a menor de todas. Vale observar também que a dispersão entre os resultados encontrados pelo PSO foi menor do que a encontrada pelo FSS.

Fazendo a análise do gráfico da Figura 15, pode-se observar a taxa de falso positivo obtida por cada algoritmo no treinamento *AdaBoost*. A menor taxa encontrada na média foi a obtida através do treinamento do GA, porém sua taxa de acerto inviabiliza que esse algoritmo seja utilizado como solução. A segunda menor taxa de acerto foi encontrada pelo PSO, que também atingiu uma melhor dispersão de resultados do que o FSS.

Ao analisar os resultados obtidos, pode-se ter o PSO como melhor algoritmo dentre os testados para o treinamento *AdaBoost* para vinte classificadores, apresentando-se como a solução mais robusta dentre as testadas.

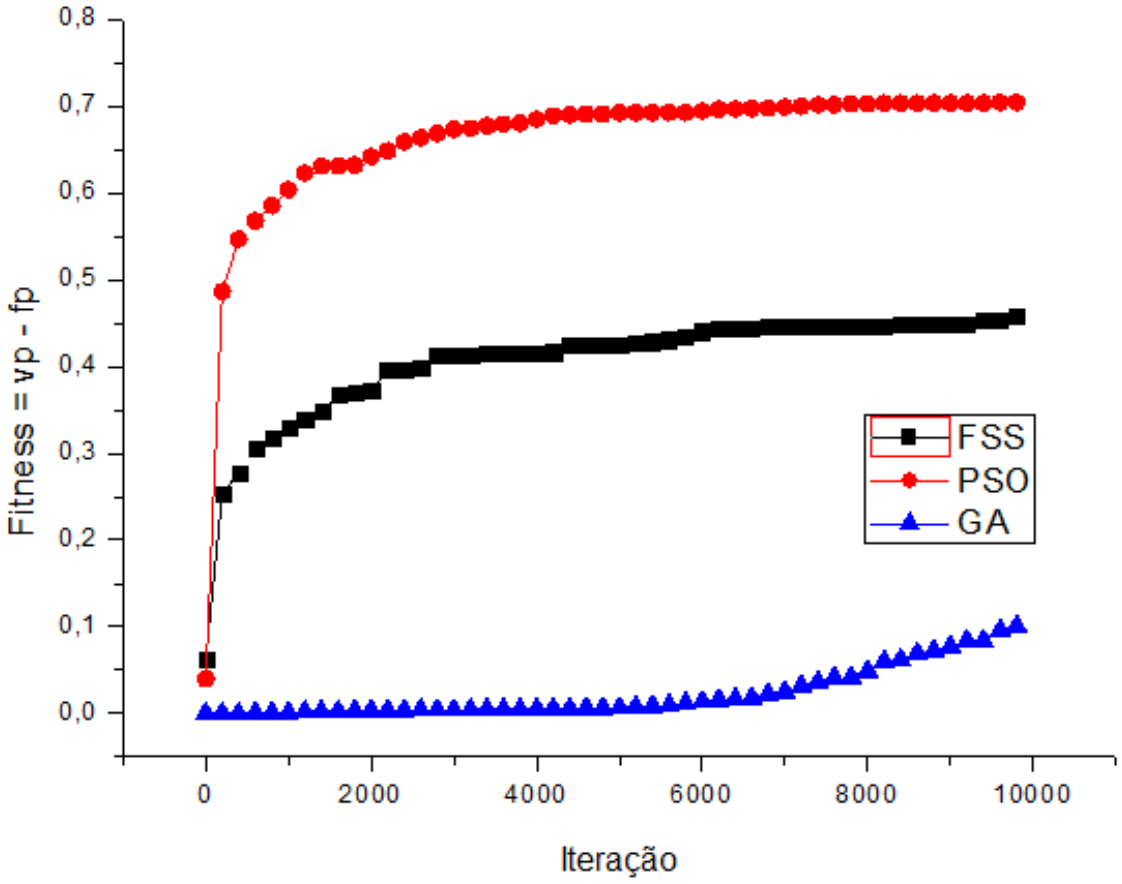

Figura 13 - Comparativo entre o *fitness* das técnicas implementadas ao longo das iterações para dez classificadores fracos.

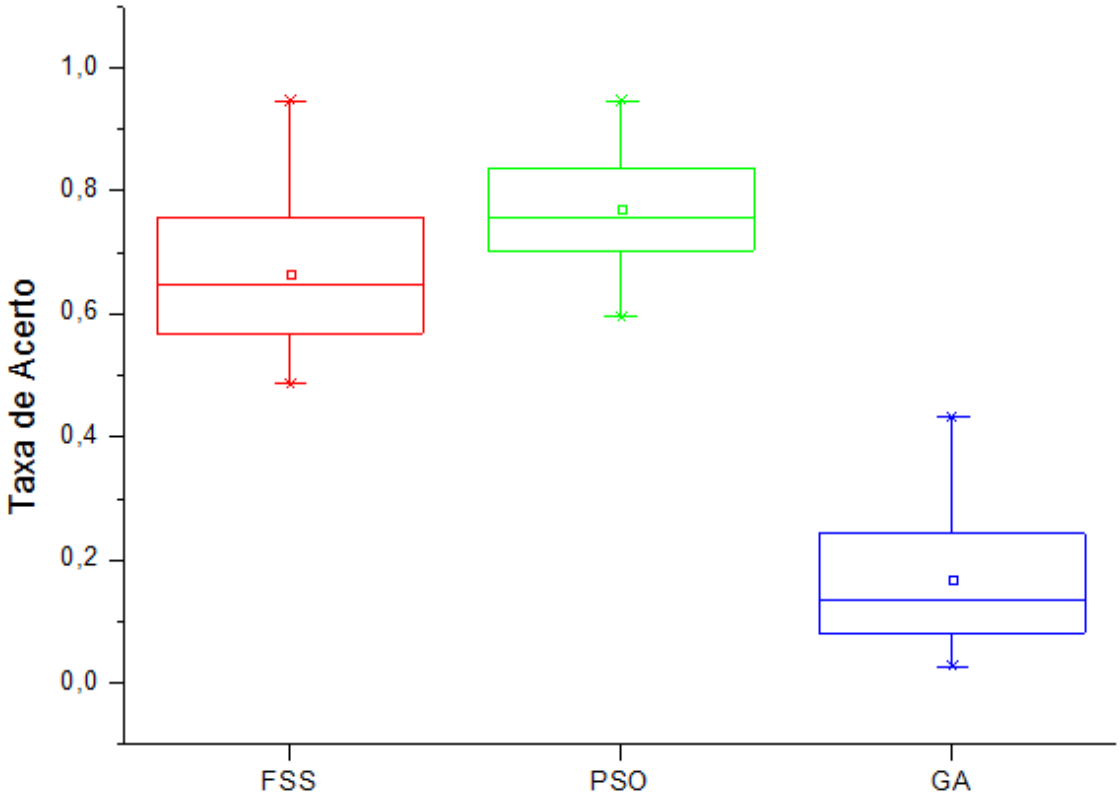

Figura 14 - Comparativo entre as taxas de acerto utilizando 20 classificadores fracos

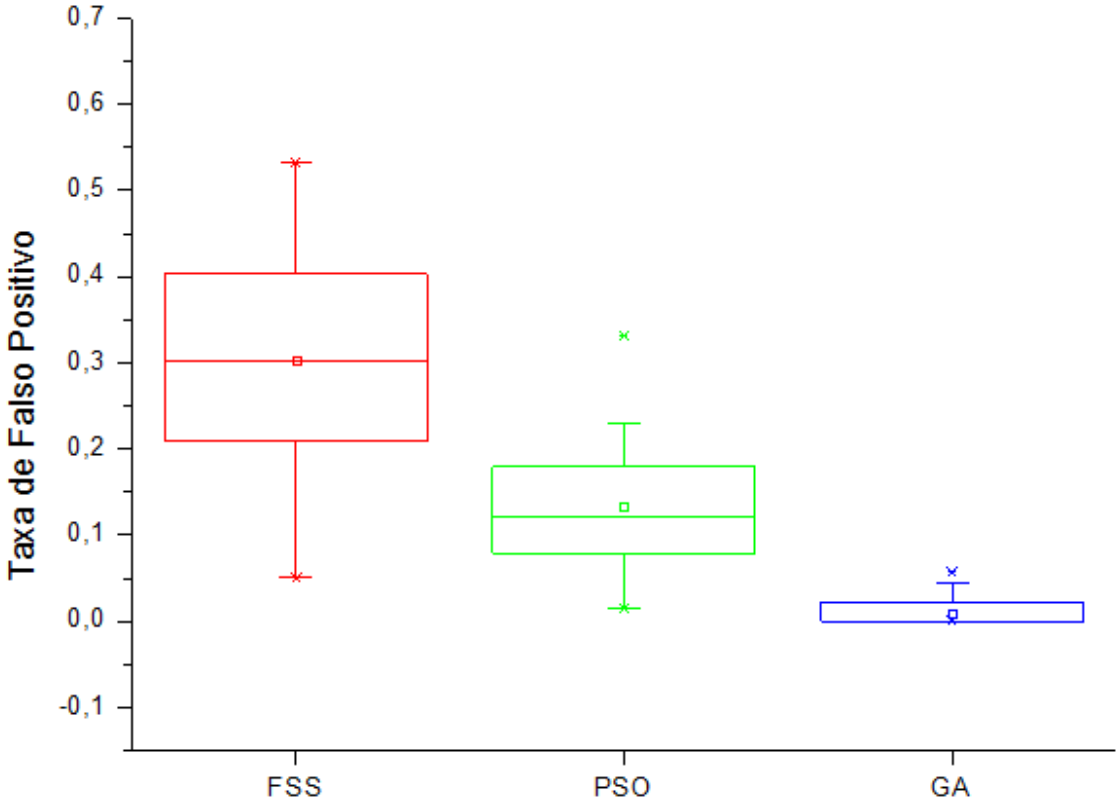

Figura 15 - Comparativo entre as taxas de falso positivo utilizando 20 classificadores fracos

5. Conclusão e trabalhos futuros

5.1. Conclusão

Os algoritmos utilizados no treinamento *AdaBoost* apresentam características distintas. O PSO é um algoritmo que possui grande capacidade de realizar busca em profundidade. O GA, por sua vez, é um algoritmo que apresenta grande capacidade de realizar busca em amplitude, enquanto o FSS tem como ponto forte a capacidade de escapar de mínimos e máximos locais.

Pode-se concluir que para tanto para cinco quanto para dez classificadores o treinamento do *AdaBoost* realizado pelo GA alcançou melhores resultados e mostrou-se mais robusto. A capacidade de realizar busca em largura proveu uma maior diversidade de soluções inicialmente, graças aos operadores de combinação e de mutação. Isto provavelmente acarreta a menor velocidade de aprendizado apresentada em todos os casos pelo GA.

Por outro lado, para um grande número de classificadores (vinte classificadores) que aumenta bastante a dimensionalidade, o aprendizado com os parâmetros utilizados foram ineficientes e precisou-se de muitas iterações para que o algoritmo fugisse de máximos locais após sua inicialização. Talvez alterando o processo de mutação, tanto aumentando sua taxa quanto permitindo que mais de um classificador sofresse mutação possa trazer melhores resultados para o caso de vinte classificadores.

Os gráficos de convergência (representados pelas Figuras 7, 10 e 13) mostram que o GA ainda apresentava tendência de crescimento na avaliação da função objetivo, diferentemente dos outros dois algoritmos, que já demonstravam certo grau de estagnação.

O PSO, por sua vez, conseguiu resultados próximos aos do GA para cinco e dez classificadores e apresentou os melhores resultados para vinte classificadores. Isso se deve a sua alta capacidade de busca em profundidade, que refinou as soluções encontradas pelo algoritmo conseguindo lidar com as altas dimensionalidades. Seus classificadores obtiveram sempre uma taxa de acerto mais altas e falso positivo mais baixas quando comparadas às taxas encontrados pelas outras técnicas e obtiveram melhores resultados no teste com vinte classificadores.

O treinamento utilizando o FSS mostrou que o algoritmo para cinco e dez classificadores obteve as maiores taxas de acerto, porém ao custo de ter uma maior variabilidade entre os resultados mostrados para cada experimento. Por outro lado não conseguiu gerar classificadores restritivos, com baixas taxas de falso positivo em nenhuma das três quantidades de classificadores testados, demonstrando incapacidade de refinamento na sua busca.

Como pôde ser observado ao longo do Capítulo 4, os resultados para cinco classificadores obtiveram uma melhor avaliação da função objetivo. Sendo assim, o GA para cinco classificadores apresentou-se como melhor solução.

5.2. Trabalhos futuros

Como trabalhos futuros podem ser feitos testes utilizando outros algoritmos como busca por colônia de abelhas (ABC, *Artificial Bee Colony*) e evolução diferencial (DE, *Differential Evolution*), além de algoritmos multiobjetivos para realizar o treinamento *AdaBoost* para usar tanto o aumento da taxa de acerto como a diminuição da taxa de falso positivo como os dois objetivos a serem atingidos. Neste trabalho a função objetivo, como já foi mencionada, foi obtida fazendo a subtração das duas taxas.

Além disso, uma investigação mais a fundo sobre o desempenho do GA para simulações com mais de dez mil iterações para verificar se a tendência de crescimento nos gráficos de convergência serão concretizadas.

Outro ponto a ser estudado é a utilização do PSO para gerar as soluções iniciais a serem utilizadas pelo GA ao longo das iterações. Uma solução composta pelo PSO inicializando por quinhentas iterações, por exemplo, e utilizando esses classificadores encontrados como entrada para o treinamento desempenhado pelo GA traria os benefícios da rapidez de aprendizado demonstrada pelo PSO com o aprendizado mais refinado demonstrado pelo GA.

6. Referências

[1] D. S. Vanni, B. Horstmann, A. M. Benjo, J. Paulo, L. Daher, S. Kanaan, and M. Sleiman, “Óxido nítrico: inibição das plaquetas e participação na formação do trombo Nitric oxide : inhibition of platelets and participation in thrombus formation,” pp. 181–189, 2007

[2] L. Arthur, C. Leite, N. Marques, S. Junior, M. S. Miranda, D. Hematologia, C. Mestre, and D. D. H. Unifesp, “Comparação Entre a Contagem de Plaquetas pelos Métodos Manual e Automatizado,” 2007.

[3] S. R. Comar, H. S. M. Danchura, and P. H. Silva, “Contagem de plaquetas: avaliação de metodologias manuais e aplicação na rotina laboratorial,” *Revista Brasileira de Hematologia e Hemoterapia*, vol. 31, no. 6, pp. 431–436, 2009

[4] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR 2001*, vol. 1, no. C, p. I–511–I–518, 2001.

[5] Firmo, A. C. A., "Classificação *AdaBoost* com Treinamento por Enxame de Partículas para Diagnóstico da Esquistossomose Mansônica no Litoral de Pernambuco," [s.n.], 2010.

[6] Engelbrecht, A. P.; Computational intelligence An introduction. [S.d.]: Wiley, 2007.

[7] C. J. a. Bastos Filho, F. B. de Lima Neto, A. J. C. C. Lins, A. I. S. Nascimento, and M. P. Lima, “A novel search algorithm based on fish school behavior,” *2008 IEEE International Conference on Systems, Man and Cybernetics*, pp. 2646–2651, Oct. 2008.

[8] D. N. O. Nascimento, "Classificação *AdaBoost* para detecção e contagem automática de plaquetas," [s.n.], 2011.

[9] P. Viola and M. Jones, “Robust Real-Time Face Detection,” *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[10] Y. Freund and R. E. Schapire, “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting,” *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[11] R. E. Schapire, Y. Freund, M. Hill, and P. Bartlett, “Boosting the margin : A new explanation for the effectiveness of voting methods,” 1997.

[12] P. S. Carbonetto, “Robust object detection using boosted learning,” *Department of Computer Science, University of British Columbia, Vancouver*, 2002.

[13] A. H. Wright, “Genetic Algorithms for Real Parameter Optimization.”

[14] R. Hinterding, “Gaussian Mutation and Self-adaption for Numeric Genetic Algorithms,” 1995

[15] H. K. Birru, “Empirical Study Of Two Classes Of Bit Variation Operators In Evolutionary Computation,” pp. 1917–1924, 1999.

[16] J. Kennedy and R. Eberhart, “Particle swarm optimization,” *Optics Express*, vol. 4, no. 3, pp. 1942–1948, 1995.