

Otimização por Colônia de Partículas

Sezimária de F.P Saramago, Jair Rocha do Prado

Faculdade de Matemática - FAMAT,
Universidade Federal de Uberlândia
38408-100, Uberlândia, MG
E-mail: Saramago @ufu.br , jair@mat.ufu.br,

1. Introdução

Problemas de otimização são caracterizados por situações em que se deseja maximizar ou minimizar uma função numérica de várias variáveis, num contexto em que podem existir restrições. Tanto as funções acima mencionadas como as restrições dependem dos valores assumidos pelas variáveis de projeto ao longo do procedimento de otimização.

Pode-se aplicar otimização em várias áreas, como por exemplo no projeto de sistemas ou componentes, planejamento e análise de operações, problemas de otimização de estruturas, otimização de forma, controle de sistemas dinâmicos.

A otimização tem como vantagens diminuir o tempo dedicado ao projeto, possibilitar o tratamento simultâneo de uma grande quantidade de variáveis e restrições de difícil visualização gráfica e/ou tabular, possibilitar a obtenção de algo melhor, obtenção de soluções não tradicionais, menor custo.

As técnicas clássicas de otimização são conhecidas à bem mais de um século, sendo utilizadas na física e na geometria, servindo-se de ferramentas associadas às equações diferenciais ao Cálculo Variacional. A sofisticação dos recursos computacionais, desenvolvidos nos últimos anos, tem motivado um grande avanço nas técnicas de otimização. Aliado ao fato de que os problemas tornam-se cada vez mais complexos.

Técnicas clássicas de otimização são confiáveis e possuem aplicações nos mais diferentes campos de engenharia e de outras ciências. Porém, estas técnicas podem apresentar algumas dificuldades numéricas e problemas de robustez relacionados com: a falta de continuidade das funções a serem otimizadas ou de suas restrições, funções não convexas, multimodalidade, existência de ruídos nas funções, necessidade de se trabalhar com valores discretos para as variáveis, existência de mínimos ou máximos locais, etc. Assim, os estudos de métodos heurísticos, com busca randômica controlada por critérios probabilísticos, reaparecem como uma forte tendência nos últimos anos, principalmente devido ao avanço dos recursos computacionais, pois um fator limitante destes métodos é a necessidade de um número elevado de avaliações da função objetivo [1].

Métodos clássicos possuem como grande vantagem, o baixo número de avaliações da função objetivo, o que faz com que tenham convergência

rápida. Contudo, estes métodos têm uma dificuldade para trabalhar com mínimos locais. Como estes métodos utilizam um único ponto do espaço de projeto e informações sobre os gradientes, ao se depararem com mínimos locais estes métodos não conseguem avançar na busca, convergindo prematuramente, sem encontrar o mínimo global.

Nos métodos de otimização natural, a função objetivo é avaliada várias vezes, sendo possível trabalhar com vários pontos ao mesmo tempo em uma iteração (população). Isto eleva o custo computacional destes métodos. Entretanto, este fato é compensado pela menor chance que estes métodos têm de serem “presos” em mínimos locais. Há claramente uma relação de compromisso estabelecida.

De forma geral, os métodos de otimização natural requerem maior esforço computacional quando comparados aos métodos clássicos, mas apresentam vantagens tais como: fácil implementação, robustez e não requerem continuidade na definição do problema [3].

Como exemplo desta classe de métodos, pode-se citar os Algoritmos Genéticos, que trabalham com técnicas de computação evolutiva, as quais modelam a evolução das espécies proposta por Darwin e operando sobre uma população de candidatos (possíveis soluções). A idéia é que a evolução da população faça com que a formação dos cromossomos dos indivíduos caminhe para o ótimo, à medida que aumenta sua função de adaptação (fitness).

O algoritmo conhecido como Colônia de Partículas (Particle Swarm Optimization), um método baseado no comportamento social de aves. A busca por alimento ou pelo ninho e a interação entre os pássaros ao longo do vôo são modelados como um mecanismo de otimização. Fazendo uma analogia, a área sobrevoada é equivalente ao espaço de projeto e encontrar o local com comida ou o ninho corresponde a encontrar o ótimo. O algoritmo é baseado em um modelo simplificado da teoria de enxames (swarm theory), através da qual os pássaros ou partículas fazem uso de suas experiências e da experiência do próprio bando para encontrarem o ninho ou alimento. As aplicações presentes na literatura evidenciam a capacidade do algoritmo na solução de diferentes problemas, bem como salientam a habilidade de trabalhar com variáveis discretas e contínuas simultaneamente.

1.1 Problema Geral de Otimização

O problema geral de otimização consiste em minimizar uma função objetivo, sujeita, ou não, a restrições de igualdade, desigualdade e restrições laterais.

A função objetivo e as funções de restrições podem ser funções lineares ou não lineares em relação às variáveis de projeto, implícitas ou explícitas, calculadas por técnicas analíticas ou numéricas.

Seja o problema geral de otimização dado por:

Minimizar :

$$f(x), x = [x_1, x_2, \dots, x_n]^T, x \in \mathcal{R}^n \quad (1)$$

Sujeito a: $g_j(x) \geq 0, j=1,2,\dots,J$

$$h_k(x) = 0, k=1,2,\dots,K \quad (2)$$

$$x_i^{(L)} \leq x \leq x_i^{(U)}, i=1,2,\dots,n$$

onde, $f(X)$ representa a função objetivo, g_j e h_k as restrições de desigualdade e de igualdade, $x_i^{(L)}$ e $x_i^{(U)}$ as restrições laterais. Todas estas funções assumem valores em \mathcal{R}^n e são, na maioria dos casos, não-lineares.

2. Otimização por Colônia de Partículas (Particle Swarm Optimization)

Otimização por colônia de partículas (PSO), é uma técnica de otimização desenvolvida na década de 90, mais precisamente em 1995, por James Kennedy e Russel Eberhart. Neste modelo é analisado algoritmos que modelam o “comportamento social” visto em várias espécies de pássaros.

Dentre vários modelos vamos estudar a técnica desenvolvida pelo biólogo Frank Heppener que é baseada no seguinte comportamento: pássaros estão dispostos aleatoriamente e estes estão a procura por alimento e um local para construir o seu ninho, eles não sabem onde está esse lugar e este é único. A indagação é qual o melhor comportamento que os pássaros terão que realizar para conseguir efetuar seu objetivo, parece mais evidente que eles sigam o pássaro que estiver mais próximo do alimento ou do ninho. Inicialmente os pássaros voam sem nenhuma orientação prévia, eles se aglomeram em bandos, até

que um consegue encontrar o ninho e atrai os que estiverem mais próximos.

Pelo fato de um pássaro encontrar o ninho a chance de os outros pássaros também encontrarem aumenta consideravelmente, isto se deve ao fato de a inteligência ser social, ou seja, o indivíduo aprende com o acerto do outro.

2.1 O algoritmo Particle Swarm Optimization

O algoritmo Particle Swarm Optimization (PSO) foi introduzido por James Kennedy e Russell Elberhart em 1995 e emergiu de experiências com algoritmos que modelam o “comportamento social” observado em muitas espécies de pássaros [5], os pássaros são chamados de partículas e durante a busca por alimento ou ninho usam de suas experiências e da experiência do bando. O PSO é um algoritmo que possui um vetor de velocidades e outro de posição, a posição de cada partícula é atualizada de acordo com a velocidade atual, o saber adquirido pela partícula e o saber adquirido pelo bando. O fluxograma mostrado na Figura 1 representa um esboço do algoritmo [2].

A posição das partículas é calculada segundo a equação:

$$x_{k+1}^i = x_k^i + v_{k+1}^i \Delta t \quad (3)$$

Onde:

x_{k+1}^i é a posição de cada partícula i na iteração $k+1$

v_{k+1}^i é o vetor de velocidade desta partícula

Δt : equivale ao espaço de tempo considerado.

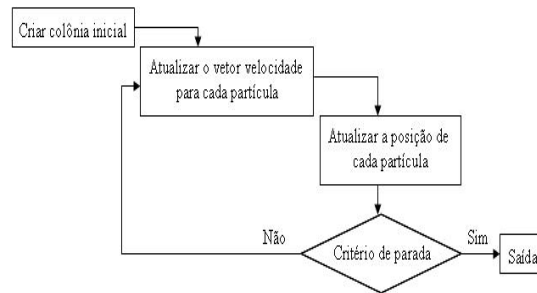


Figura 1 – Fluxograma para o algoritmo PSO básico

O vetor de velocidade é atualizado conforme a equação:

$$v_{k+1}^i = wv_k^i + c_1 r_1 \frac{(p^i - x_k^i)}{\Delta t} + c_2 r_2 \frac{(p_k^s - x_k^i)}{\Delta t} \quad (4)$$

Considerando que, v_k^i é a velocidade atual da partícula;

r_1, r_2 são números aleatórios entre 0 e 1;

p^i é a melhor posição encontrada pela partícula i e

p_k^s é a melhor posição do bando na iteração k .

v_s^i – velocidade próxima ao ótimo da colônia

v_p^i – velocidade próxima ao ótimo da partícula

p^s – colônia ótima

p^i – partícula ótima

● - posição atual

○ - posição próxima

2.2 Colônia Inicial

O cálculo da velocidade necessita, ainda, de alguns parâmetros dependentes do problema, que são: a inércia da partícula (w), que controla a capacidade de exploração do algoritmo, ou seja, um valor alto facilita um comportamento mais global, enquanto um valor baixo facilita um comportamento mais local [3], e os dois parâmetros de confiança c_1 e c_2 que indicam o quanto uma partícula confia em si (c_1), e no bando (c_2). A Figura 2 mostra a aplicação da equação anterior, considerando duas partículas se deslocando em um espaço de projeto bidimensional.

Os parâmetros de confiança e de inércia devem ser ajustados de acordo com o problema, pois são utilizadas para a atualização do vetor velocidades. Alguns autores propõem que sejam adotados $c_1 = c_2 = 2$ e $0.7 < w < 1.4$. Sugere-se, também, a adoção de valores diferentes para c_1 e c_2 desde que satisfaçam $c_1 + c_2 = 4$.

A inércia pode ser atualizada de forma iterativa pela expressão:

A inicialização da população de colônia normalmente é obtida com as partículas dispostas aleatoriamente sobre o espaço de projeto, cada uma possui um vetor de velocidade aleatório inicial. As equações a seguir mostram como são obtidos a posição e o vetor de velocidades iniciais.

$$x_0^i = x_{\min} + r_1 (x_{\max} - x_{\min}) \quad (6)$$

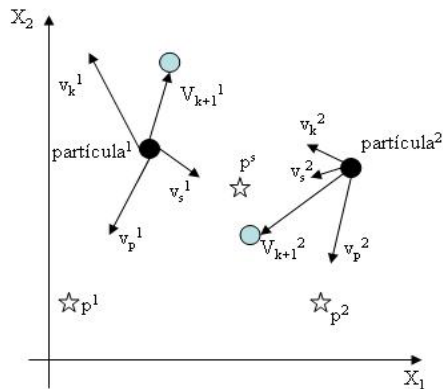
$$v_0^i = \frac{x_{\min} + r_2 (x_{\max} - x_{\min})}{\Delta t} \quad (7)$$

Onde,

r_1 e r_2 são números aleatórios entre 0 e 1;

x_{\min} é o limite inferior das restrições laterais para as variáveis de projeto;

x_{\max} é o limite superior das restrições laterais para as variáveis de projeto.



$$w_{new} = f_w w_{old} \quad (5)$$

Figura 2: Vetor de velocidades em ação

Considerando o fator de redução, f_w uma constante entre 0 e 1. São usados neste trabalho, a inércia constante $w_0 = 0.729$ e $c_1 = c_2 = 2$.

Onde:

2.3 Otimização com restrições

Os algoritmos evolutivos e PSO, por tratarem-se de algoritmos naturais, não trabalham diretamente com restrições. Uma estratégia para se fazer com que estes algoritmos manipulem restrições, é utilização de funções de penalidade quadrática estendida.

Assim, defini-se uma função pseudo-objetivo definida $\Psi(x)$:

$$\Psi(x) = f(x) + rp \sum_{i=1}^m \max[0, g_i(x)]^2 \quad (8)$$

Sendo,

$f(x)$ a função objetivo original;

rp um fator de penalidade (de ordem variável segundo o tipo de problema);

$g_i(x)$ o conjunto de todas as restrições (com violações para $g_i(x) > 0$);

Quando há restrições nos problemas de otimização, as partículas que desrespeitam alguma restrição se enquadram em um grupo que merecem um tratamento

especial, esse tratamento se inicia pelo cálculo do novo vetor de velocidade, dado pela seguinte equação:

$$v_{k+1}^i = c_1 r_1 \frac{(p^i - x_k^i)}{\Delta t} + c_2 r_2 \frac{(p_k^s - x_k^i)}{\Delta t} \quad (9)$$

Observe que a Equação (9) não leva em consideração a informação do vetor de velocidade na iteração anterior para o novo cálculo do vetor de velocidade, isto se deve ao fato de a partícula estar “se divergindo” em direção a uma violação.

Na maioria das vezes este novo vetor de velocidades se destinará a uma região viável e a partícula sai da restrição.

2.4 Variáveis de Projeto

Discretas / Inteiras

O PSO é um algoritmo contínuo, contrapondo os Algoritmos Genéticos que primeiramente eram destinados a variáveis discretas. Porém, o PSO pode ser muito eficiente na resolução de problemas com variáveis discretas, desde que sejam feitas algumas modificações no algoritmo, por exemplo a posição de cada partícula é arredondada para o valor inteiro mais próximo logo em seguida a aplicação da Equação (3) ou da Equação (6).

3. Simulação Numérica

A seguir para a realização das simulações numéricas foram utilizados os programas GAOT para Algoritmos Genéticos e um código desenvolvido em Matlab para o PSO.

Exemplo 1:

a) $f(x) = \exp(x) * \text{sen}(x)$; $0 < x < 9,3$;

GAOT: $F_{\min} = -172,6409$;
 $x = 5,4978$;

PSO: $F_{\min} = -172,6409$;
 $x = 5,4978$;

b) $f(x) = \exp(x) * \text{sen}(x)$; $0 < x < 9.3$;

GAOT: $F_{\max} = 3995,0$;
 $x = 8,6394$;

PSO: $F_{\max} = 3995,0$;
 $x = 8,6394$;

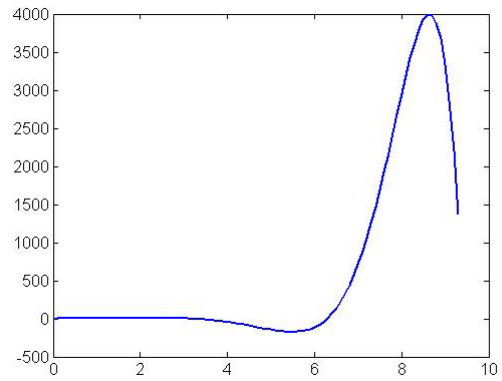


Figura 3 – Representação gráfica de $y = \exp(x) * \text{sen}(x)$

Exemplo 2:

a) $f(x) = (x_1 + 2)^2 + (x_2 - 1)^2$; $-10 < x_1 < 10$; $-10 < x_2 < 10$;

GAOT: $F_{\min} = 0$;
 $x = [-2 \ 1]$;

PSO: $F_{\min} = 0$;
 $x = [-2 \ 1]$;

b) $f(x) = (x_1 + 2)^2 + (x_2 - 1)^2$; $-10 < x_1 < 10$; $-10 < x_2 < 10$;

GAOT: $F_{\max} = 265$;
 $x = [10 \ -10]$;

PSO: $F_{\max} = 265$;
 $x = [10 \ -10]$;

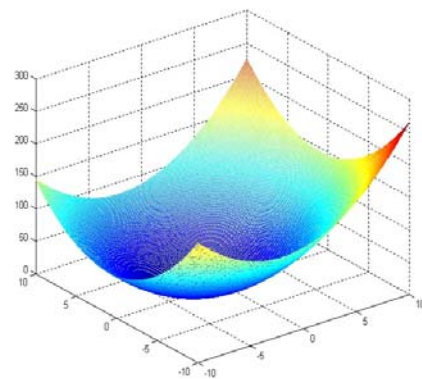


Figura 4 – Representação gráfica de $y = (x_1 + 2)^2 + (x_2 - 2)^2$

Exemplo 3:

a) $f(x) = (x_1 + 2)^2 + (x_2 - 1)^2 + x_3$; $-10 < x_1 < 10$; $-10 < x_2 < 10$; $-10 < x_3 < 10$

GAOT: Fmin = -10;
 $x = [-2 \ 1 \ -10]$;

PSO: Fmin = -10;
 $x = [-2 \ 1 \ -10]$;

b) $f(x) = (x_1 + 2)^2 + (x_2 - 1)^2 + x_3$; $-10 < x_1 < 10$; $-10 < x_2 < 10$; $-10 < x_3 < 10$

GAOT: Fmáx = 275;
 $x = [10 \ -10 \ 10]$;

PSO: Fmáx = 275;
 $x = [10 \ -10 \ 10]$;

Veja a seguir alguns exemplos aplicados em funções matemáticas utilizando o código computacional PSO e comparados a outros algoritmos evolutivos.

1 - $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2$; $-5.12 \leq x_i \leq 5.12$;

Parâmetros utilizados no programa:

TipoF = 3, estr = 9, NP = 6, F=0.95, CR=0.5, $\epsilon = 1.e-6$, itermax = 100, k=20

Resultados: Fmin = 5.1833e-008, I = 8, $x = (0.9999, 1.0008)$

PSO: Fmin = 1.0651 * e-011;
 $x = (1.0000, 1.0000)$

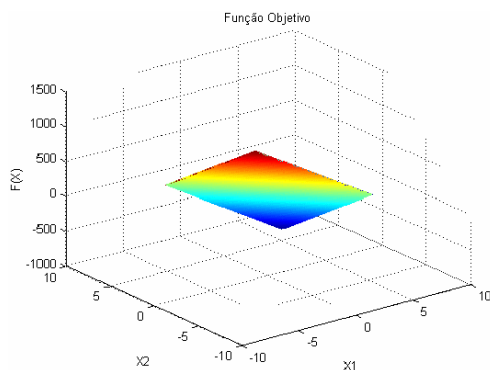


Figura 5 - Representação gráfica de $y = (x_1 - 1)^2 + (x_2 - 1)^2$

2 - $f(x_i) = \sum_{i=1}^3 x_i^2$ - Primeira função de Jong

(esfera), $-5.12 \leq x_i \leq 5.12$;

TipoF = 3, estr = 9, NP = 10, F = 0.5, CR = 0.3, $\epsilon = 1.e-6$, itermax = 100

Fmin = 5.1833e-008, I = 8, $x = (0.9999, 1.0008)$

Fmin = 2.5628e-008, I = 14, $x = 1.0e-003 * (0.3769, -0.3678, 0.7613)$

Fmin = 9.8810e-008, I = 14, $x = 1.0e-003 * (0.7336, -0.2945, -0.4629)$

Fmin = 5.2698e-008, I = 4, $x = 1.0e-003 * (-0.4543, -0.0603, -0.0786)$

Fmin = 3.9860e-008, I = 13, $x = 1.0e-003 * (0.0246, 0.1211, -0.1937)$

PSO: Fmin = 7.4955 * e-011; $x = 1.0 * e-004 * (-0.0745, -0.2425, 0.0538)$

3 - $f(x_1, x_2) = 100 * (x_1^2 - x_2)^2 + (1 - x_1)^2$ - Segunda função de Jong, $-2.048 \leq x_i \leq 2.048$;

TipoF = 3, estr = 7, NP = 6, F = 0.95, CR = 0.5, $\epsilon = 1.e-6$, itermax = 100, nf = 600

Fmin = 7.4639e-007, I = 13, $x = (1.00001, 1.0007)$

Mesmos parâmetros, porém estr = 9

Fmin = 3.9849e-009, I = 8, $x = (0.9998, 0.9996)$, nf = 480

PSO: Fmin = 2.5632e-007 $x = (0.9998, 0.9995)$,

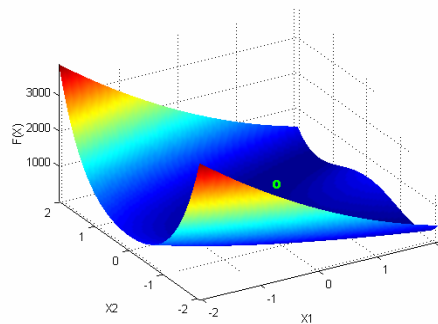


Figura 6 - Representação gráfica de $y = 100 * (x_1^2 - x_2)^2 + (1 - x_1)^2$

4 - $f(x_1, x_2) = x_1 * \text{sen}(4x_1) + 1.1 * x_2 * \text{sen}(2x_2)$ - SA e AG (Saramago);

SA: $x = (9.0388, 8.6682)$, Fmin = -18.557, %contorno de 8 a 10

AG: $x = (9.0390, 8.6682)$, Fmin = -185547, %idem

ED: $x = (9.0310, 8.6679)$, Fmin = -18.5501, nf = 30, tipoF = 4, estr = 9, x entre 8 e 10

TipoF = 2, estr = 7, NP = 15, F = 0.95, CR = 0.3, $\epsilon = 1.e-6$, itermax = 500, k = 5000

Fmin = -18.5443, I = 244, $x = (8.9983, 9.0576)$

PSO: Fmin = -18.5547; $x = (9.0390, 8.6682)$; $-10 \leq x_i \leq 10$

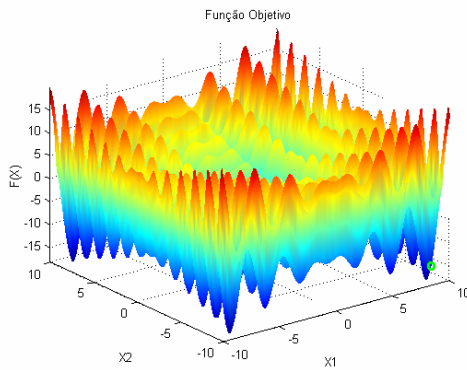


Figura 7 – Representação gráfica de $y = x_1 * \text{sen}(4x_1) + 1.1 * x_2 * \text{sen}(2x_2)$

$$5 - f(x_1, x_2) = [(x_1^2 - 10)^2 + (x_2 - 10)^2]^{1/10}, -10 \leq x_i \leq 10;$$

TipoF = 3, estr = 7, NP = 10, F = 0.95, CR = 0.5, $\varepsilon = 1.e-6$, itermax = 100, k = 20, nf = 20

Fmin = 0, I = 1, x = (10, 10) %

vários testes mostrou repetibilidade

Mesmos parâmetros porém bounds = [-40,40] % repetibilidade

Fmin = 0.0311, I = 20, x = (3.1623, 10), nf=1000

PSO: Fmin = 0.0448 x = (3.1623,10)

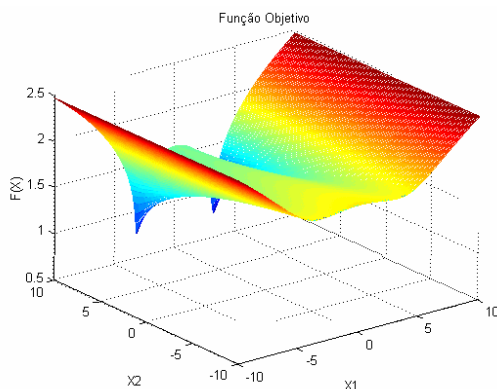


Figura 8 – Representação gráfica de $y = [(x_1^2 - 10)^2 + (x_2 - 10)^2]^{1/10}$

Conclusão

Este trabalho apresenta um estudo sobre algoritmos evolutivos, considerando duas técnicas desenvolvidas recentemente: otimização por colônia de partículas (particle swarm) e algoritmos genéticos.

Através de simulações numéricas aplicadas a problemas simples, pode-se observar que as duas técnicas convergem para os mesmos resultados.

Além disso, observa-se que a otimização por colônia de partículas trabalha com um tamanho de população bastante reduzido, portanto seu esforço computacional é pequeno. Este fato incentiva pesquisas futuras, onde esta técnica será aplicada a problemas complexos que requerem muitas avaliações da função objetivo.

Referências Bibliográficas

[1] H.P. Schwefel E L Taylor, H.P. “Evolution and Optimum Seeking”, John Wiley & Sons Inc, United States of America, pp. 87-88, 1994.

[2] Rojas, J. E., Viana,, F.A.C., Rade, D. A. and Rojas, J. E., Viana,Steffen Jr, V., “Force identification of mechanical systems by using particle swarm optimization”. In Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Albany, New York, Aug 30-01 Sept 2004.

[3]Venter, G. And Sobieszczanski-Sobieski, J., “Particle Swarm Optimization”, Proceedings of the 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Denver, CO, Vol. AIAA-2002-1235, April 22-25 2002.

[4]Kennedy, J. and Eberhart, R. C., “Particle Swarm Optimization”, Proceedings of the 1995 IEEE International Conference on Neural Networks, Perth, Australia, 1995, pp. 1942-1948.

[5]Pomeroy, P., “An Introduction to Particle Swarm Optimization”, <http://www.adaptiveview.com>, [15 Setembro de 2003].