SPOLM 2008

ISSN 2175-6295 Rio de Janeiro- Brasil, 05 e 06 de agosto de 2008.

UMA NOVA METAHEURÍSTICA BASEADA EM ALGORITMO DE COLISÃO DE MÚLTIPLAS PARTÍCULAS

Eduardo Fávero Pacheco da Luz
Pós-graduação em Computação Aplicada (CAP)
Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1.075 – Jardim da Granja – CEP: 12227-010 – São José dos Campos/SP
eduardo.luz@lac.inpe.br

José Carlos Becceneri
Laboratório Associado de Computação e Matemática Aplicada (LAC)
Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1.075 – Jardim da Granja – CEP: 12227-010 – São José dos Campos/SP
becce@lac.inpe.br

Haroldo Fraga de Campos Velho
Laboratório Associado de Computação e Matemática Aplicada (LAC)
Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1.075 – Jardim da Granja – CEP: 12227-010 – São José dos Campos/SP
haroldo@lac.inpe.br

RESUMO

Este trabalho apresenta um novo conceito e os resultados iniciais de uma nova metaheurística baseada na implementação de um algoritmo de otimização de funções baseado na colisão de múltiplas partículas, com viabilização através de um ambiente computacional de alto desempenho. Os resultados obtidos na aplicação do *Multi-Particle Collision Algorithm* (M-PCA) nas funções de teste de Easom, Schekel, Rosenbrock e Griewank revelam resultados animadores, que junto com a garantia de um menor esforço computacional garantem a aplicabilidade do método.

Palavras-Chaves: Otimização; Colisão de partículas; Múltiplas partículas; Computação de alto desempenho.

Abstract

This work presents a new concept and the initial results of a new metaheuristics based on the implementation of a minimization/maximization functions algorithm based on the collision of multiple particles, viable through a high performance computing environment. The obtained results for the application of Multi-Particle Collision Algorithm (M-PCA) over Easom, Schekelm Rosenbrock and Griewank test functions reveals outstanding results, which together with the assurance of less computational efforts ensure the applicability of the method.

Keywords: Optimization; Particle collision; Multiple particles; High performance computing.

1. INTRODUÇÃO

Ao longo dos tempos, cientistas tem investigado e desenvolvido novos métodos para resolução de problemas de otimização [1]. Estes novos métodos podem ser baseados em fenômenos físicos (*Simulated Annealing*, *Particle Colission Algorithm*), fenômenos biológicos (*Genetic Algorithm*), comportamento animal (*Particle Swarm Optimization*) e até mesmo em padrões de crescimento vegetal (*Invasive Weed Optimization*).

Em trabalhos anteriores, um método baseado em comportamento animal (PSO) obteve grande sucesso na resolução de um problema inverso, expresso como um problema de otimização ([2],[3]). Assim, ficou comprovada a aplicabilidade destas técnicas para a resolução de problemas de aplicação prática e de grande relevância social.

Após esse resultado, novas técnicas inspiradas na natureza foram estudadas, e a técnica baseada no algoritmo de colisão de partículas (PCA) recebeu maior atenção por apresentar características que poderiam ser exploradas com o intuito de melhorar a resposta obtida.

O algoritmo original implementa este comportamento para uma única partícula, fazendo uma busca serializada no domínio, sendo esta busca baseada na técnica apresentada por [4]. Porém esta busca pode ser melhorada através da utilização de mais de uma partícula, para que, de maneira colaborativa, o espaço de buscas seja mais bem explorado, com o objetivo de evitar a convergência para um ótimo local.

Assim, este trabalho apresenta um novo algoritmo para otimização de funções baseado na expansão da idéia original, adotando um novo paradigma com múltiplas partículas para a condução da otimização, que com a exploração do espaço de buscas sendo conduzida de maneira mais eficiente, acaba por garantir uma significante melhoria, que levou a resolução de problemas que o algoritmo original não se mostrou capaz.

2. OTIMIZAÇÃO

A teoria da otimização é o ramo da matemática que engloba o estudo quantitativo do ótimo e os métodos usados para encontrar o ótimo. Esta teoria posta em prática é definida pela coleção de técnicas, métodos, procedimentos e algoritmos que podem ser usados para a localização do ótimo [5].

Problemas de otimização tem por objetivo encontrar a melhor combinação dentre um conjunto de variáveis para maximizar ou minimizar uma função, definida como sendo uma função objetivo ou função custo. Os problemas de otimização podem ser classificados como: problemas de otimização contínua (cujas variáveis assumem valores reais ou contínuas); problemas de otimização combinatória ou discreta (cujas variáveis assumem valores discretos ou inteiros); e problemas de otimização mista (com variáveis inteiras e contínuas ao mesmo tempo) [6].

O melhor método para a localização do mínimo de uma função depende fortemente da natureza da função em estudo, sendo que, em linhas gerais, duas classes de algoritmos podem ser usadas: os algoritmos de minimização local, que dado um ponto em um subdomínio da função, eles tentam, e quase sempre conseguem, localizar o ponto mais baixo deste subdomínio (quase sempre algorimos dados por métodos determinísticos); os algoritmos de minimização global, que tentam minimizar a função encontrando o ponto mais baixo dentre todos os subdomínio existentes no espaço total (exemplos incluem algoritmos estocásticos baseados em metaheurísticas).

Os métodos determinísticos usados na otimização de problemas irrestritos podem ser classificados em três tipos principais, sendo que esta classificação leva em consideração a quantidade de informação necessária a sua operação ([7], [8]):

1) Métodos de ordem zero: somente o valor da função objetivo é utilizado. Ex.: Método das direções conjugadas de Powell. Obs.: Usado em funções altamente

não-lineares;

2) Métodos de primeira ordem: usam o valor da função objetivo e de suas derivadas (gradientes) em relação às variáveis de projeto . Ex.: Máxima descida;
3) Métodos de segunda ordem: utilizam o valor da função objetivo, de suas derivadas e da matriz Hessiana. Ex.: Métodos de Newton e Quasi-Newton.

Os principais métodos estocásticos para a otimização são aqueles baseados em heurísticas e metaheurísticas. O termo heurística, baseado na palavra grega heuriskein, significa “descobrir” e sua atual conotação induz a existência de um método “que, baseado na experiência ou julgamento, parece conduzir a uma boa solução de um problema, mas que não garante produzir uma solução ótima” [6].

Dentre as principais metaheurísticas, principalmente aquelas fortemente baseadas em fenômenos naturais, podemos citar o *Simulated Annealing* (SA), *Genetic Algorihtm* (GA), *Particle Swarm Optimization* (PSO), *Ant Colony Optimization* (ACO), dentre outras.

3. ALGORITMO DE COLISÃO DE PARTÍCULAS

O algoritmo de colisão de particulas (Particle Colision Algorithm) foi desenvolvido pelo Dr. Wagner F. Sacco ([9], [10], [11], [12]) com grande inspiração no algoritmo de recozimento simulado (*Simulated Annealing*) apresentado por [13].

A inspiração tirada do mundo real usada para a construção do algoritmo tem suas raízes na física das reações de colisão de partículas nucleares, com grande ênfase nos comportamentos de espalhamento (*scattering*) e absorção (*absorption*). Seu uso tem sido comprovadamente eficaz em diversos casos de otimização, tanto de problemas de teste (benchmark) quanto em problemas de aplicações reais.

O algoritmo de PCA pode ser descrito através do seguinte pseudo-código, sendo que as funções relativas ao seu funcionamento seguem abaixo do trecho de código principal:

```
//Trecho principal
Gera uma solução inicial Old_Config
Best_Fitness = Fitness(Old_Config)
Para n=0 até # de iterações
  Perturbation()
  Se Fitness(New_Config) > Fitness(Old_Config)
    Se Fitness(New_Config) > Best_Fitness
      Best_Fitness = Fitness(New_Config)
    Fim-Se
    Old_Config = New_Config
    Exploration()
  Senão
    Scattering()
  Fim-Se
Fim-Para

//Função para exploração
Exploration()
  Para n=0 até # de iterações
    Small_Perturbation()
    Se Fitness(New_Config) > Fitness(Old_Config)
      Old_Config = New_Config
    Fim-Se
  Fim-Para
Retorna

//Função para espalhamento
Scattering()
```

```
  p_scattering = 1 – Fitness(New_Config) / Best_Fitness
  Se p_scattering > Random(0,1)
    Old_Config = Solução aleatória
  Senão
    Exploration();
  Fim-Se
Retorna

//Função que implementa a perturbação da solução
Perturbation()
  Para i=0 até (Dimensões-1)
    Superior = Limite_superior[i]
    Inferior = Limite_inferior[i]
    Aleatorio = Random(0,1)
    New_Config[i] = Old_Config[i] + ((Superior – Old_Config[i] *
Aleatorio) – ((Old_Config[i] – Inferior)*(1-Aleatorio))
    Se New_Config[i] > Superior
      New_Config[i] = Limite_superior[i]
    Senão
      Se New_Config[i] < Inferior
        New_Config[i] = Limite_inferior[i]
      Fim-Se
    Fim-Se
  Fim-Para
Retorna

//Função que implementa uma pequena perturbação na solução
Small_Perturbation()
  Para i=0 até (Dimensoes-1)
    Superior = Random(1.0,1.2) * Old_Config[i]
    Se Superior > Limite_superior[i]
      Superior = Limite_superior[i]
    Fim-Se
    Inferior = Random(0.8,1.0) * Old_Config[i]
    Se Inferior < Limite_inferior[i]
      Inferior = Limite_inferior[i]
    Fim-Se
    Aleatorio = Random(0,1)
    New_Config[i] = Old_Config[i] + ((Superior –
Old_Config[i])*Aleatorio)-((Old_Config[i] – Inferior)*(1-Aleatorio))
  Fim-Para
Retorna
```

A estrutura de funcionamento do PCA leva em consideração a escolha de uma solução inicial (`Old_Config`), que então sofre uma modificação, definida de um modo estocástico (pela função `Perturbation()`), para a construção de uma nova solução (`New_Config`). As qualidades das duas soluções são calculadas (função `Fitness()`) e imediatamente comparadas, e então a nova solução pode ser aceita ou não, dependendo de sua performance frente à solução anterior.

Caso esta nova solução não seja aceita, o esquema do tipo Metropolis é posto em prática (pela função `Scattering()`), ou seja, uma solução candidata com uma performance inferior à atual é selecionada com uma dada probabilidade. E é justamente este esquema de seleção estocástica que vem a garantir a não-convergência prematura para mínimos locais.

Para garantir que as perturbações impostas ao algoritmo não levem à construção de uma solução na qual um de seus pontos esteja fora do espaço válido de buscas, uma verificação destes limites é feita constantemente para evitar este contratempo, tal como o efetuado dentro das funções `Perturbation()` e `Small_Perturbation()`.

Nos termos da engenharia nuclear, se a qualidade de uma nova solução que está sendo proposta é melhor do que a qualidade da solução anterior, esta nova partícula (ou

posição) é absorvida e o próximo passo é constituído pela realização de uma busca local, caracterizado pela ativação da função `Small_Perturbation()`, com o objetivo de verificar a existência de uma possível solução ainda melhor dentro de uma determinada vizinhança deste novo ponto.

Caso o novo ponto não se prove melhor, o esquema do algoritmo de Metropolis entra em ação novamente, e dada uma certa probabilidade, a partícula que representa a solução pode ser espalhada (reinicializada, de uma certa maneira) através da função `Scattering()`.

O PCA é um algoritmo robusto e de fácil implementação. Uma de suas principais vantagens é a quantidade de parâmetros livres necessários à sua inicialização. O usuário só precisa fornecer a quantidade de iterações sobre a qual o algoritmo definirá o seu critério de parada, logo, somente um parâmetro livre deve ser ajustado, ou definido para iniciar o algoritmo. Esta é uma grande vantagem quando em comparação com outras metaheurísticas, onde um grande número de parâmetros requer um melhor ajuste inicial destes parâmetros. Segundo a literatura, [10], 10^4~10^6 iterações tem se mostrado um valor suficiente.

4. ALGORITMO DE COLISÃO DE MÚLTIPLAS PARTÍCULAS

Baseado no algoritmo apresentado no início deste capítulo, este trabalho propõe, implementa e testa uma nova versão do algoritmo de *Particle Collision Algorithm*, que introduz o conceito e o uso de multi-partículas, possibilitado por sua implementação em um ambiente de computação de alto desempenho para agilizar o processo e ao mesmo tempo, abrir um maior leque de exploração do espaço de buscas.

A implementação desta nova versão baseada em múltiplas partículas estende o conceito básico da aplicação de uma única partícula para a exploração da totalidade do espaço de buscas em um dado tempo total de iterações. Nesta nova versão, é utilizado um conjunto de n partículas explorando, de maneira independente, porém relativamente colaborativa, este mesmo espaço de buscas.

Além, é claro, do fato de nos aproveitarmos da maior capacidade computacional que um ambiente de alto desempenho pode vir a fornecer, o uso de mais de uma partícula pode garantir, com maior grau de certeza, uma convergência em menor espaço de tempo, O que pode diminuir ainda mais a possibilidade de que o algoritmo fique preso em mínimos locais.

4.1. COMPUTAÇÃO DE ALTO DESEMPENHO

Ambientes de alto desempenho são possibilitados através do uso de computação de alto desempenho (*High Performance Computing* – HPC), termo que reflete o uso de supercomputadores (paralelos) e/ou clusters de computadores.

O processo de computação, que pode ser definido como a execução de uma seqüência de instruções em um conjunto de dados, pode ser agilizado quando o uso de um ambiente de alto desempenho é posto em prática. Neste ponto, a computação em um ambiente paralelo se consiste de uma ou mais tarefas que são executadas de maneira concorrente, sendo que o número destas tarefas pode variar durante a execução do programa.

Em termos gerais, os sistemas computacionais que possibilitam ao usuário um ambiente de computação em termos de teraflops são designados computadores-HPC.

A taxionomia apresentada por [14] visa classificar os computadores e programas capacitados a operações de alta velocidade em ambientes de alto desempenho. As possíveis classes são:

- *Single Instruction Stream – Single Data Stream* (SISD);
- *Single Instruction Stream – Multiple Data Stream* (SIMD);
- *Multiple Instruction Stream – Single Data Stream* (MISD);
- *Multiple Instruction Stream – Multiple Data Stream* (MIMD);

4.2. MULTI-PARTICLE COLLISION ALGORITHM (M-PCA)

O algoritmo proposto para o *Multi-Particle Collision Algorithm* (M-PCA), é, em sua essência, exatamente o mesmo do PCA canônico. Porém a introdução de n partículas efetuando a busca no mesmo espaço de soluções leva a necessidade da implementação de um mecanismo de comunicação indireta entre as partículas, para que uma coordenação das atividades seja capacitada.

```
//Trecho principal
Gera uma solução inicial Old_Config
Best_Fitness = Fitness(Old_Config)
Para n=0 até # de partículas
  Para n=0 até # de iterações
    Perturbation()
    Se Fitness(New_Config) > Fitness(Old_Config)
      Se Fitness(New_Config) > Best_Fitness
          Best_Fitness := Fitness(New_Config)
      Fim-Se
      Old_Config = New_Config
      Exploration()
    Senão
      Scattering()
    Fim-Se
  Fim-Para
Fim-Para
```

Esta coordenação foi viabilizada através da implementação de uma técnica de blackboard, onde o Best_Fitness, o melhor resultado obtido pela partícula do PCA agora seja atualizado de maneira constante, com a melhor posição dentre todas as partículas envolvidas no processo de busca.

Ou seja, as partículas trabalham de maneira independente no processo de busca da solução, porém no momento em que alguma partícula identifica uma nova melhor posição, esta partícula "escreve" no blackboard do M-PCA para que as outras partículas tomem conhecimento deste novo valor, e então os seus critérios de busca, tais como a definição de um novo pscattering, variável em uso na função Scattering() e que leva em consideração o melhor fitness da melhor solução atual, sejam baseados neste valor, definido como sendo o melhor mínimo atualmente em uso, e de conhecimento disseminado, pelo conjunto de partículas.

De maneira semelhando ao PCA, o M-PCA só possui um parâmetro livre, o número de iterações que é responsável pelo critério de parada do algoritmo. A redução de tempo proporcionada pelo M-PCA advém da capacidade de dividir o total de iterações imposta ao algoritmo pelo número de partículas que atuarão de maneira conjunta no espaço de buscas, ou seja, para uma inicialização do PCA com 10^5 iterações, uma instância do M-PCA com 10 partículas terá o número de iterações reduzidas para 104, caso 100 partículas sejam usadas, 10^3 iterações serão suficientes para um número equivalentes de avaliações no espaço de buscas.

4.2.1. Complexidade

Estimar a complexidade de um novo algoritmo é um passo crucial para a validação e até mesmo valorização da técnica que está sendo apresentada. Se para a execução de uma dada tarefa, vários algoritmos de mesma precisão estão disponíveis, a escolha lógica recai, em geral, para aquele de menor complexidade computacional ([15], [16]).

A análise de complexidade de um algoritmo tem por objetivo a avaliação da eficiência deste algoritmo em termos de tempo ou espaço computacional requeridos, [16].

Quando fazemos uma análise da complexidade relativa à execução do algoritmo

PCA canônico, podemos verificar, segundo o trecho principal de seu código fonte, que temos uma complexidade $O(n^2)$, dada a existência de dois laços aninhados.

O algoritmo de múltiplas partículas M-PCA força a inserção de um novo laço, em um nível superior aos laços existentes, o que leva a construção de um novo trecho principal, que possui complexidade $O(n^3)$.

No momento da paralelização do M-PCA se este laço for distribuído para n processadores, a complexidade do algoritmo retorna para $O(n^2)$. Esta redução da complexidade é possível através da implementação em um ambiente de alto desempenho, com características de um esquema SIMD, tal como o apresentado na seção sobre computação de alto desempenho.

Fazendo a análise baseada na classificação apresentada por [17], o algoritmo de M-PCA é implementado em um sistema de software que tem por base a biblioteca MPI e viabilizado através de um hardware com arquitetura de um sistema distribuído.

A implementação do M-PCA também pode ser dada em um ambiente SISD, o equivalente a um computador pessoal comum em implementação serial, porém neste caso a complexidade relacionada à execução do algoritmo é cúbica, $O(n^3)$, o que faz com que a principal vantagem desta variante, que pode ser sintetizada como sendo a aplicação colaborativa de n instâncias do algoritmo original levando a uma exploração mais minuciosa do espaço de buscas em um mesmo período de tempo, não se aplique.

Podemos então fazer uma análise de *speedup*, que é definida como a razão entre o tempo de execução do algoritmo seqüencial (T_s) e o tempo de execução do algoritmo paralelo (T_p), seguida de uma análise de eficiência (E_p), que é definida pela razão entre o *speedup* e o número de processadores (p), com o objetivo de verificar o quanto o paralelismo foi explorado no algoritmo. Um bom resultado implica em $E_p <= 1$ ([17]).

Adotando a resolução da função de Easom, que será descrita com maiores detalhes na seção de Resultados, para fins de comparação, obtemos os seguintes valores para o M-PCA: $S_p = 4.447$ e $E_p = 0.4447$, que comprovam a eficiência do algoritmo paralelizado

5. RESULTADOS

As funções usadas para teste foram as funções de Easom, Shekel, Rosenbrock e Griewank, sendo que Easom, Rosenbrock e Griewank são funções para minimização e a função de Shekel é de maximização. Para o PCA, 10^5 iterações foram definidas para todas as funções teste. O M-PCA foi executado com 10 partículas, cada uma fazendo 10^4 iterações.

A função de Easom é expressa pela equação abaixo:

$$f(x) = -\cos(x_1)\cos(x_2)\exp\left(-(x_1-\pi)^2-(x_2-\pi)^2\right) \quad (1)$$

Sendo que o espaço de buscas é definido entre $-100 \le x_i \le 100,\ i = 1,2$. Esta função possui um mínimo global, que é localizado em $x^* = (\pi, \pi)$, com $f(x^*) = -1$.

Os resultados comparados entre o PCA e M-PCA são apresentados na tabela abaixo:

Tabela 1: Comparativo de resultados para a função de Easom entre o PCA e M-PCA.

	PCA	*M-PCA*
Tempo	19.37*s*	4.29*s*
*x**	(3.14,3.14)	(3.14,3.14)
f(*x**)	–1.00	–1.00

A segunda função testada é a função de Shekel. Uma função para maximização, que tem sua expressão pela equação abaixo:

$$f(x) = 500 - \frac{1}{0.002 + \sum_{i=0}^{24} \frac{1}{1 + i + \left(x_1 - a(i)\right)^6 + \left(x_2 - b(i)\right)^6}} \tag{2}$$

Em que,

$$\begin{aligned} a(i) &= 16\left[(i \bmod 5) - 2\right] \\ b(i) &= 16\left[(i/5) - 2\right] \end{aligned} \tag{3}$$

Estando concentrada em $-65.536 \le x_i \le 65.536$ e possuindo máximo em $x^* = (-32, -32)$ com $f(x^*) = 499.002$.

A tabela de resultados comparativos entre PCA e M-PCA segue abaixo:

Tabela 2: Comparativo de resultados para a função de Shekel entre o PCA e M-PCA.

	PCA	*M-PCA*
Tempo	3099.53*s*	113.56*s*
*x**	(–32.03,–31.91)	(–32.02,–31.94)
f(*x**)	499.0019	499.0019

A terceira função é a de Rosenbrock, uma função de minimização expressa pela equação abaixo e com espaço de buscas descrito como segue:

$$f(x) = 100\left(x_2 - x_1^{\,2}\right)^2 + (1 - x_1)^2 \tag{4}$$

Seu espaço de buscas está concentrado em $-2.048 \le x_i \le 2.048$ possuindo mínimo global em $x^* = (1,1)$ com $f(x^*) = 0$.

Os resultados obtidos são apresentados na tabela abaixo:

Tabela 3: Comparativo de resultados para a função de Rosenbrock entre o PCA e M-PCA.

	PCA	*M-PCA*
Tempo	1492.23*s*	21.21*s*
*x**	(1.00, 1.00)	(1.00, 1.00)
f(*x**)	5×10^{-12}	9×10^{-12}

A quarta e última função é a de Griewank, uma função de otimização que possui uma topologia peculiar, sendo a que melhor demonstra a superioridade do M-PCA sobre sua versão original. A equação que a descreve se encontra abaixo e seu domínio é apresentado nas abaixo:

$$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{5}$$

Estando concentrada em $-600 \le x_i \le 600$ e n representa a quantidade de dimensões sendo exploradas. O mínimo global está localizado no ponto $x^* = (0, \mathrm{K}, 0)$ com $f(x^*) = 0$.

Os resultados comparativos entre o PCA e M-PCA são apresentados na tabela abaixo:

Tabela 4: Comparativo de resultados para a função de Griewank entre o PCA e M-PCA.

	PCA	*M-PCA*
Tempo	2392.05*s*	32.03*s*
x^*	(-3.14,4.43)	$(-1.82\times10^{-8},-3.25\times10^{-8})$
$f(x^*)$	7.39×10^{-3}	3.33×10^{-16}

6. CONSIDERAÇÕES FINAIS

O novo algoritmo de colisão de múltiplas partículas apresentou resultados satisfatórios para a sua aplicação em funções de benchmark, inclusive obtendo boas soluções para funções as quais o algoritmo na sua versão original não se demonstrou adequado.

Estes resultados, que foram viabilizados através da implementação e uso do novo algoritmo em um ambiente computacional de alto desempenho, demonstram a grande capacidade de aplicação deste novo algoritmo para diversos tipos de problemas de otimização.

Trabalhos futuros podem incluir uma maior investigação sobre a real influência da expressão de exponencial truncada atualmente em uso na função de espalhamento (scatering), podendo ser substituída por uma função densidade de probabilidade de Cauchy, [18], ou até mesmo por uma função associada à distribuição de Tsallis, que segundo a literatura vem apresentando bons resultados quando associada à resolução de problemas por metaheurísticas, [19].

Maiores experimentos também podem ser feitos no tocante ao comportamento geral das partículas: para a implementação do M-PCA, um comportamento colaborativo foi selecionado onde as partículas exploram juntas e indiscriminadamente o espaço de buscas, porém uma alternativa a este comportamento pode ser sintetizado pela competição entre partículas ou grupo de partículas.

Logo, o algoritmo de colisão de múltiplas partículas (M-PCA) pode ser visto como uma alternativa viável para a resolução de problemas de otimização quando o usuário tem a sua disponibilidade um ambiente de computação de alto desempenho.

7. AGRADECIMENTOS

Os autores gostariam de agradecer ao Dr. Wagner Figueiredo Sacco, da Universidade do Estado do Rio de Janeiro, criador do PCA, pelas discussões sobre seu algoritmo e o envio de seus códigos originais para comparação com os implementados; à equipe responsável pelo C-PAD/INPE; e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo financiamento parcial deste trabalho através de bolsa.

8. REFERÊNCIAS BIBLIOGRÁFICAS

[1] BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*. Vol. 35, N. 3. pp. 268-308, 2003.

[2] LUZ, E. F. P. *Estimação de fonte de poluição atmosférica usando otimização por enxame de partículas*. Dissertação (Mestrado em Computação Aplicada). pp. 86. São José dos Campos: INPE: 2007.

[3] LUZ, E. F. P.; CAMPOS VELHO, H. F.; BECCENERI, J. C.; ROBERTI, D. R. Estimating atmospheric area source strengh through particle swarm optimization. Inverse Problems, Design and Optimization, IPDO 2007, Florida, *Proceedings…*, Florida: IPDO, 2007. (CD-ROM).

[4] METROPOLIS, N.; ROSENBLUTH, A. W.; ROSENBLUTH, M. N.; TELLER, A. H., Equation of state calculations by fast computing machines, *Journal of Chemical Physics*, Vol. 21, Iss. 6, pp. 1087-1092, 1953. Disponível em: http://link.aip.org/link/?JCPSA6/21/1087/1, Acesso em: 22 abr. 2008.

[5] ANTONIOU, A.; LU, W-S. *Practical optimization: algorithms and engineering applications*. New York: Springer, 2007.

[6] BECCENERI, J. C. Meta-Heurísticas e Otimização Combinatória: Aplicações em Problemas Ambientais. pp. 65-81. *Computação e Matemática Aplicada às Ciências e Tecnologias Espaciais*. ROSA, R. R. e SILVA, J. D. S. (Org.) 1 ed. São José dos Campos: INPE, 2008.

[7] NELLI SILVA, E. C. *Otimização aplicada ao projeto de sistemas mecânicos*. (Notas de aula). Departamento de Engenharia Mecatrônica e Sistemas Mecânicos. Escola Politécnica – USP. 2007.

[8] GOLDBARG, M. C.; LUNA, H. P. L. *Otimização combinatória e programação linear: modelos e algoritmos*. 2 ed. Rio de Janeiro: Elsevier, 2005.

[9] SACCO, W. F.; ALVES FILHO, H.; PEREIRA, C. M. N. A. Cost-based optimization of a nuclear reactor core design: a preliminary model. 2007 International Nuclear Atlantic Conference, INAC 2007, *Procedings…*, Santos: ABEN, 2007.

[10] SACCO, W. F.; DE OLIVEIRA, C. R. E. A new stochastic optimization algorithm based on a particle collision metaheuristic. 6th World Congress of Structural and Multidisciplinary Optimizaion, *Procedings…*, Rio de Janeiro: WCSMO, 2005.

[11] SACCO, W. F.; DE OLIVEIRA, C. R. E.; PEREIRA, C. M. N. A. Two stochastic optimization algorithms applied to nuclear reactor core design. *Progress in Nuclear Energy*. Vol. 48, pp. 525-539, 2006.

[12] SACCO, W. F.; LAPA, C. M. F.; PEREIRA, C. M. N. A.; ALVES FILHO, H. A metropolis algorithm applied to a nuclear Power plant auxiliary feedwater system surveillance tests policy optimization. *Progress in Nuclear Energy*. N. 50, pp. 15-21, 2008.

[13] KIRKPATRICK, S.; GELATT, C. D.;VECCHI, M. P. Optimization by Simulated Annealing. *Science*, Vol. 220, Number 4598, pp. 671-680, 1983. Disponível em: http://www.cs.virginia.edu/cs432/documents/sa-1983.pdf. Acesso em: 22 abr. 2008.

[14] FLYNN, M. J. Very High-Speed Computing Systems. *Proceedings of the IEEE*. Vol. 54. Iss. 12. pp. 1901- 1909. 1966. Disponível em: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1447203. Acesso em: 24 abr. 2008.

[15] ZIVIANI, N. *Projeto de algoritmos: com implementações em PASCAL e C*. 2ª Ed. São Paulo: Pioneira Thomson Learning, 2004.

[16] SZWARCFITER, J. L.; MARKENZON, L. *Estruturas de dados e seus algoritmos*. 2ª Ed. Rev. Rio de Janeiro: LTC Editora, 1994.

[17] SAMBATTI, S. B. M. *Diferentes estratégias de paralelização de um algoritmo genético epidêmico aplicadas na solução de problemas inversos*. Dissertação (Mestrado em Computação Aplicada). 70 p. São José dos Campos: INPE, 2004.

[18] WANG, Z.; HANSON, J.V. Codebook optimization by Cauchy annealing. Global Telecommunications Conference, 1993, Houston, TX, USA, *Technical Program Conference Record*. Vol. 3, pp. 1974-1978, 1993, ISBN: 0-7803-0917-0.

[19] HANSMANN, U. H. E. Simulated annealing with Tsallis weights: a numerical comparison. *Physica A*, Vol. 242, N. 1, pp. 250-257(8), 1997.