PSOView: Simulador Web para o Algoritmo de Otimização Global Particle Swarm Optimization

Ricardo Grunitzki[1], Fernando dos Santos[1]

[1]Departamento de Sistemas de Informação – Univ. Estado de Santa Catarina (UDESC)
Rua Dr. Getúlio Vargas, 2.822 – CEP: 89.140-000 – Bela Vista – Ibirama – SC – Brasil

ricardo.grunitzki@gmail.com, fernando.santos@udesc.br

Abstract. *Since the creation of the particle swarm optimization algorithm, it is remarkable the increasing number of publications proposing different parameter settings or implementation strategies. However there is a big difficulty of representing the effects of such proposals during the optimization process. This difficulty is due the few existing tools are not flexible enough and require the installation of additional resources to run. This paper presents the development of a web simulator for the global optimization algorithm particle swarm optimization. With the simulator it is possible to follow the behavior of the algorithm in the main optimization problems or in any function defined by the user.*

Resumo. *Desde a criação do algoritmo particle swarm optimization, é notável o crescente número de publicações propondo diferentes configurações de parâmetros ou estratégias de implementação. Porém, há uma grande dificuldade em representar os efeitos de tais propostas durante o processo de otimização, já que as poucas ferramentas existentes não são flexíveis o bastante e requerem a instalação de recursos adicionais para executar. Este trabalho apresenta o desenvolvimento de um simulador web para o algoritmo de otimização global particle swarm optimization. Com o simulador pode-se acompanhar o comportamento do algoritmo nos principais problemas de otimização ou a partir de uma função qualquer definida pelo usuário.*

1. Introdução

Na natureza existem várias espécies que se beneficiam da sociabilidade, pois a vida em grupos sociais aumenta a probabilidade de acasalamento, facilita a caça e coleta de alimentos, reduz a probabilidade de ataque por predadores e permite a divisão do trabalho [Zuben e Attux 2008]. Baseados nas vantagens que certos indivíduos possuem ao viver coletivamente no mundo real, pesquisadores desenvolveram ferramentas computacionais para a solução de problemas e estratégias de coordenação e controle, ao que chamam de *Swarm Intelligence*.

Técnicas de inteligência computacional com fundamentos baseados no *Swarm Intelligence* realizam uma analogia à natureza, onde indivíduos integrantes de uma população interagem localmente uns com os outros e também com seu meio ambiente. Tal princípio fundamenta o algoritmo *particle swarm optimization* ou PSO. Proposto inicialmente por Eberhart e Kennedy em 1995, o PSO é um algoritmo estocástico com

características heurísticas de busca randômica, que relaciona conceitos da psicologia social e inteligência computacional [Kennedy e Eberhart 2001].

Desde então, muitos esforços tem sido investidos na obtenção do melhor entendimento das propriedades de convergência do PSO. Esses estudos concentram-se principalmente na compreensão dos parâmetros básicos de controle do PSO, como os coeficientes de aceleração, peso inercial, tamanho de população, topologia de vizinhança e distribuição da população inicial [Van Den Bergh e Engelbrecht 2006].

Em função de o algoritmo apresentar um conceito simples, sendo de fácil implementação e possuindo poucos parâmetros de ajuste, têm sido encontradas inúmeras aplicações para o PSO. Dentre essas aplicações destacam-se a resolução de problemas com restrições, problemas de minimização ou maximização, problemas com múltiplas soluções, monitoramento dinâmico e otimização de pesos e arquiteturas de redes neurais [Shi 2004].

A necessidade de aprimoramento das estratégias de algoritmos estocásticos com heurísticas inteligentes está baseada no melhor conhecimento a respeito dos processos, o que provoca o surgimento de modelos fenomenológicos cada vez mais complexos, assim como do acesso às informações, que devem ser correlacionadas e interpretadas. Nesse contexto a utilização de algoritmos determinísticos é inviabilizada, diante de sua dificuldade quanto à convergência a uma solução global e dependência dos critérios de inicialização [Thomas e Reed 2009].

As propostas de aprimoramento nos algoritmos estocásticos devem ser experimentadas para validar seus benefícios. Neste sentido, é importante o uso de algum simulador, onde seja possível avaliar o comportamento em diferentes situações. Ferramentas de simulação consistem na utilização de técnicas matemáticas em computadores, com o intuito de imitar o funcionamento de praticamente qualquer tipo de operação do mundo real. Permitem a experimentação da realidade através de um modelo e assim, avaliar como as variáveis irão se comportar no sistema idealizado [Harrell *et al.* 2002].

O simulador proposto neste trabalho oferece um ambiente de testes flexível, acessível, independente de plataforma e recursos adicionais para avaliar o comportamento e desempenho do PSO nos mais diversos problemas de minimização. O simulador, chamado PSOView, é executado via web, através de qualquer navegador compatível com HTML5. Além disso, traz nove famosos casos de otimização implementados: Ackley, Alpine, Griewank, Rastrigin, Rosenbrock, Schaffer's f6, Schwefel, Sphere e Tripod. Além das funções já implementadas, o simulador ainda permite a criação de casos de testes personalizáveis por meio de uma equação matemática. O simulador também permite alterar tanto os parâmetros da função teste quanto do algoritmo PSO.

Este trabalho está organizado da seguinte maneira. A seção 2 apresenta a fundamentação necessária sobre o algoritmo PSO, assim como a sua aplicação em problemas de otimização. A seção 3 apresenta os dois principais trabalhos correlatos. A seção 4 apresenta o desenvolvimento do sistema PSOView, sua apresentação e comparação com trabalhos correlatos. Por fim, a seção 5 apresenta as conclusões e trabalhos futuros.

2. Particle Swarm Optimization

O algoritmo inteligente *particle swarm optimization* é uma técnica de computação evolucionária desenvolvida por Eberhart e Kennedy em 1995, inspirada no comportamento social de espécies biológicas como aves e peixes. É uma técnica baseada em inteligência computacional que não é afetada pelo tamanho ou não linearidade do problema, podendo convergir para uma solução ótima em muitos problemas onde métodos mais analíticos falham na convergência [Delvalle *et al.* 2008].

O PSO utiliza uma população chamada de enxame, onde cada indivíduo dentro do enxame é denominado partícula. Segundo Jiao, Lian e Gu (2008), uma partícula *i* em uma iteração *k* se desloca através do espaço solução com dois atributos: (1) a posição atual dentro de um espaço de busca *N*-dimensional $X_i^k = \left(x_1^k, ..., x_n^k, ... x_N^k\right)$ do problema, com $x_n^{min} \le x_n^k \le x_n^{max}$ para todo $n \in [1, N]$, onde x_n^{min} e x_n^{max} são os limites da coordenada *n*; (2) sua velocidade, representada vetorialmente por $V_i^k = \left(v_1^k, ..., v_n^k, ..., v_N^k\right)$ nesse mesmo espaço *N*-Dimensional do problema.

A cada iteração a velocidade e posição de todas as partículas são atualizadas de acordo com dois melhores valores encontrados durante a busca. O primeiro é o melhor valor encontrado pela partícula até o momento, chamado *pbest*. Outro melhor valor que é encontrado pelo PSO é o melhor valor encontrado até o momento por qualquer indivíduo da população, este melhor valor global é chamado *gbest*. Quando uma partícula se torna parte de uma vizinhança topológica, o melhor valor encontrado pela vizinhança é um valor local, chamado *lbest*. Segundo Jiao, Lian e Gu (2008) após encontrar os dois melhores valores, a posição e velocidade das partículas são obtidas pelas equações 1 e 2:

$$V_i\ (k+1) = wV_i(k) + c_1 r_1 \left(P_i(k) - X_i(k)\right) + c_2 r_2 \left(P_g(k) - X_i(k)\right) \quad (1)$$

$$X_i(k+1) = X_i(k) + V_I(k+1) \quad (2)$$

Onde r_1 e r_2, são números gerados aleatoriamente no intervalo entre [0,1] e c_1 e c_2 são respectivamente chamados de parâmetro cognitivo e social. O termo $c_1 r_1 \left(P_i(k) - X_i(k)\right)$ representa a distância entre a partícula *i* e sua melhor posição até a *k*-ésima interação. Este termo dispersa a busca em várias regiões do espaço solução de maneira a encontrar o mínimo global do problema. Já o termo $c_2 r_2 \left(P_g(k) - X_i(k)\right)$ representa a distância entre a partícula *i* e a melhor posição encontrada pela população até a *k*-ésima iteração. Por fim, o parâmetro *w*, introduzido por Shi e Eberhart (1998) tem como finalidade melhorar as habilidades de exploração do algoritmo no âmbito de busca, reduzindo a importância do controle da velocidade máxima [Poli *et al.* 2007].

No estudo de algoritmos de otimização é comum a realização de comparações entre diferentes algoritmos usando uma série de funções teste. As características das funções teste são as mais diversas, no entanto, em síntese a maior parte delas pode ser implementada no espaço *n* dimensional, exigindo do algoritmo a estimação de *n* parâmetros. Além disso, muitas delas são multimodais o que dificulta a otimização pelo excesso de mínimos locais, tornando-as eficientes para avaliar o desempenho dos algoritmos de otimização nas mais diversas situações.

A função de Rosenbrock, definida por $\sum_{i=0}^{n-1} 100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2$, é um clássico problema de otimização utilizada para avaliar algoritmos de otimização. Inicialmente proposta por Rosenbrock (1960) e posteriormente generalizada para *n* variáveis. Esta função tem como característica a presença de muitos mínimos locais e uma suave variação nas vizinhanças do mínimo global, tornando o processo de busca árduo para os algoritmos de otimização. A função de Rosenbrock geralmente é definida pela literatura com o espaço de busca $-100 \le x_i \le 100$, para $n = 30$. Como demonstra a Figura 1, a função apresenta mínimo global em zero, quando $x_n = (1,1,...)$. Além de Rosenbrock, são exemplos de funções testes: Ackley, Alpine, Griewank, Rastrigin, Schaffer's f6, Schwefel, Sphere e Tripod.

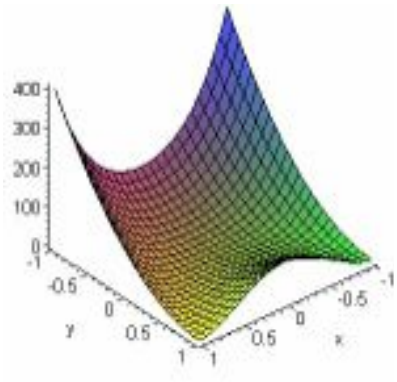


Figura 1. Representação gráfica da função de Rosenbrock em três dimensões

3. Trabalhos Correlatos

Atualmente existem poucos trabalhos desenvolvidos voltados para a visualização do processo de otimização do PSO. Nesta seção são apresentadas as duas principais ferramentas relacionadas ao assunto. São elas: PSOVis e Particle Swarm Optimization Visualization.

3.1 PSOVis

O PsoVis é uma ferramenta para a visualização do processo de otimização do algoritmo inteligente PSO. Seu objetivo é ajudar a entender o funcionamento do PSO em geral, permitindo variar os parâmetros do algoritmo e acompanhar seu comportamento de forma visual [Psovis 2005].

A aplicação possui cinco funções teste pré-definidas, sendo elas: Rastrigin, Rosenbrock, Griewank, Sphere e Schaffer f6. Apesar de não permitir a inclusão de novas funções o sistema permite alterar o espaço solução das existentes. A Figura 2 apresenta a interface visual do PsoVis [Psovis 2005].

De acordo com Psovis (2005), a ferramenta permite alterar diversos parâmetros do PSO, sendo eles: peso inercial inicial e final, coeficiente cognitivo e social, velocidade máxima, número de partículas, número e número máximo de iterações. Como configurações de função, o PsoVis permite exibir o mínimo global da função, definir o intervalo de atualização entre as iterações além de interromper e pausar o

processo de otimização. Por fim, a cada iteração os resultados são exibidos na área "*Optimization Progress*", indicado pela letra A na Figura 2.

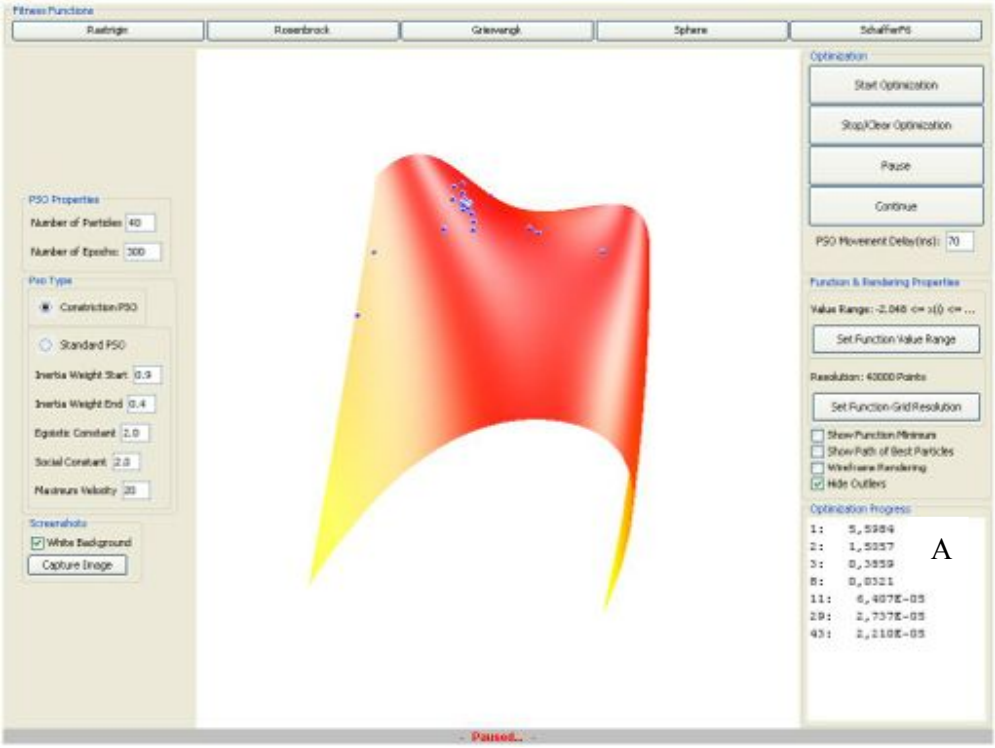


Figura 2. Applet PsoVis minimizando a função de Rosenbrock [PSOVIS 2005].

O laboratório ModLab da Goethe University Frankfurt[1], mantém o PsoVis. O laboratório também é responsável por desenvolver ferramentas de software para aplicação em bio e quimioinformática. A aplicação está disponível na web em forma de *applet* no portal do laboratório. Uma limitação desta ferramenta é que necessita instalação do Java Runtime Environment - JRE e da API Java 3D no computador do usuário.

3.2 Particle Swarm Optimization Visualization

O *Particle Swarm Optimization Visualization* ou PSO *Visualization* é uma *applet* Java que demonstra visualmente um enxame de partículas a procura de um valor máximo em uma paisagem 3D [Pso Visualization 2004]. Esta ferramenta requer JRE instalado para executar. A Figura 3 apresenta a interface gráfica da aplicação.

A cada execução, a *applet* gera um novo ambiente de busca aleatório tridimensional, onde um enxame fixo de doze partículas se movimenta em busca do máximo global, ao encontrá-lo a execução é interrompida. O enxame ainda pode ser divido em até quatro vizinhanças topológicas. A execução pode ser efetuada de uma só vez ou passo a passo, porém, o número máximo de iterações é fixado em 100, e caso o PSO não encontre uma solução a execução é interrompida.

[1] www.uni-frankfurt.de/english/

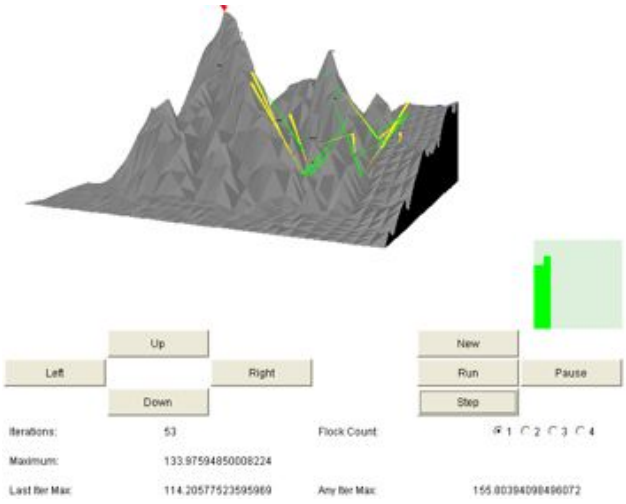


Figura 3. *Particle Swarm Optimization Visualization* maximizando um espaço tridimensional [PSO Visualization 2004]

A visualização das partículas no espaço de busca é simulada por diferentes cores que exibem a posição atual e anterior das partículas. Vetores entre as partículas representam o caminho por ela percorrido entre a anterior e atual iteração. Os botões *Up*, *Down*, *Left* e *Right* movimentam o espaço de busca (não é permitido mover através do *mouse*). Por fim, abaixo dos botões de movimentação são apresentados os resultados da otimização com número máximo de iterações, máximo global da função, máximo global encontrado pelo PSO e um campo informando se o máximo foi encontrado ou não.

Atualmente o PSO *Visualization* está hospedado e mantido pela empresa Project Computing[2]. A última atualização foi realizada em 2004 e o código fonte está disponível junto à *applet* [Pso Visualization 2004].

Como pode-se verificar, estas ferramentas possuem algumas limitações. Ambas funcionam bem nos principais navegadores, Chrome, Firefox, Internet Explorer, Safari e Opera. Porém, necessitam do JRE instalado na máquina para funcionar. O PSOVis por sua vez, ainda necessita da API Java 3D. Quanto às funções teste, nenhuma das ferramentas permite a criação de casos de testes diferentes dos disponibilizados pelas ferramentas.

4. Simulador PSOView

Esta seção apresenta os principais requisitos, a arquitetura, implementação e técnicas utilizadas na construção do simulador. Ao fim desta seção apresentam-se a operacionalização do simulador PSOView e sua comparação com trabalhos correlatos.

2 http://www.projectcomputing.com/resources/psovis/index.html

4.1 Requisitos do Sistema

O simulador proposto neste trabalho deve oferecer um ambiente de testes completo, flexível e multi-idioma, para avaliar o desempenho do PSO nas mais diversas situações. Deve ainda possibilitar o ajuste dos parâmetros do algoritmo e função teste. Quanto às funções teste, deve-se possibilitar selecionar uma função pré-definida, ou construir uma nova a partir de uma expressão matemática. O sistema terá as seguintes funções implementadas: Ackley, Alpine, Griewank, Rastrigin, Rosenbrock, Schaffer's f6, Schwefel, Sphere e Tripod.

O sistema deve ser executado diretamente em um navegador web sem a necessidade de instalação de recursos ou bibliotecas adicionais. A visualização do gráfico, deve permitir ao usuário sua interação com o sistema durante todo o processo de otimização, a fim, de melhor adequar a sua visualização. Ao fim do processo, o simulador deve fornecer ao usuário os principais resultados da otimização.

4.2 Arquitetura e implementação do Sistema

Para melhor organizar o desenvolvimento do simulador, utilizou-se arquitetura de desenvolvimento em camadas. O modelo de classes do simulador foi disposto em oito pacotes distintos. A Figura 4 apresenta o diagrama de pacotes de classes com as respectivas dependências entre os pacotes.

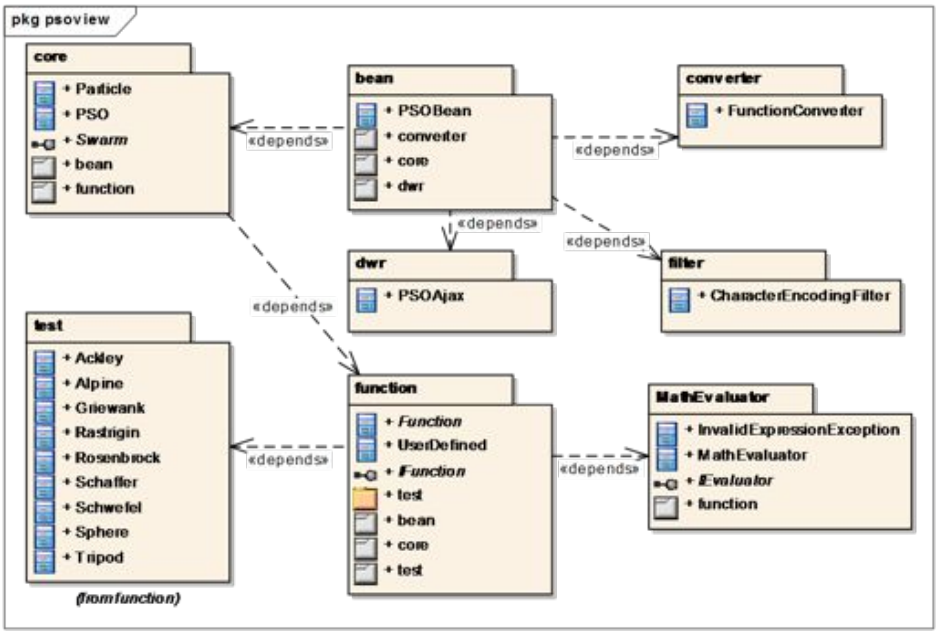


Figura 4. Diagrama de pacotes de classe

O pacote *function*, contém as classes e a interface que tornam a estrutura da aplicação flexível para criar funções teste, sendo elas definidas por meio de uma expressão ou não. O pacote *MathEvaluator* contém as classes responsáveis por calcular o valor de uma expressão matemática. Todas as classes do pacote *MathEvaluator* correspondem às classes da biblioteca MathEvaluator [Graphing 2008]. O pacote *core* armazena todas as classes responsáveis por implementar a estrutura do algoritmo PSO. Além do pacote *core*, a classe *PSOBean* do pacote *bean* se comunica com outros dois

pacotes, são eles: *converter* e *dwr*. A classe *FunctionConverter* do pacote *converter* é responsável por fazer a conversão dos *Objects* do tipo *Function* para *String* e vice e versa. Este conversor é utilizado pelo *PSOBean* quando os objetos *Function* são transmitidos para as páginas XHTML.

Finalmente a classe *PSOAjax* do pacote *dwr* possui todos os métodos responsáveis por executar requisições AJAX síncronas e assíncronas da classe *PSOBean* e arquivos JavaScript localizados no cliente. Esta implementação foi necessária para permitir a comunicação entre o Javascript e servidor, durante o processo de construção do gráfico e atualização da posição das partículas.

No instante em que a otimização é iniciada, a cada atualização do enxame de partículas, a classe observável *PSO* notifica seu observador *PSOViewBean* que há atualizações disponíveis. Este por sua vez reenvia as atualizações para a classe *PSOAjax*, a qual irá enviar uma requisição AJAX reverso ao navegador, com as novas posições do enxame. O resultado deste processo pode ser observado na Figura 7.

Observa-se que as partículas são representadas por esferas de cor rosa, com efeito gradiente interno branco. A cada atualização, função e partícula são redesenhadas, porém, este processo é tão rápido que fica imperceptível ao usuário, passando a impressão de que as partículas estão literalmente se movimentando no gráfico, enquanto tudo é redesenhado. O código fonte e a especificação completa do simulador estão disponíveis no seguinte endereço: https://code.google.com/p/psoview.

4.3 Técnicas Utilizadas

O simulador foi desenvolvido em linguagem de programação Java, com auxílio do ambiente de desenvolvimento integrado NetBeans 7.2.1 e JDK 1.7. O sistema operacional utilizado foi Windows 7.

A interface gráfica foi desenvolvida com o *framework* Java Server Faces – JSF e PrimeFaces 3.4, uma biblioteca de componentes ricos para JSF. Utilizou-se como tema para *layout* o componente *hot-sneaks* versão 1.0.8 do PrimeFaces [Prime Teknoloji 2012]. Para executar as requisições AJAX reverso e AJAX efetuadas por arquivos Javascript, utilizou-se a biblioteca *Direct Web Remoting* - DWR versão 3.1RC1 [Direct 2012]. A aplicação foi desenvolvida e testada para rodar nos servidores GlassFish versão 3.1 e TomCat versão 7.0.27.0.

Para construção gráfica das funções utilizou-se como base a biblioteca Javascript Graph3D. Esta biblioteca é mantida por [Chap 2012] e sofreu adaptações para se adequar ao problema proposto. O cálculo das expressões matemáticas foi feito através da biblioteca *Math Evaluator*. Como ambiente de testes utilizou-se os principais navegadores, sendo eles: Google Chrome versão 22.0.1, Mozilla Firefox 15.0.1, Internet Explorer 9, Opera 12.02 e Safari 6.

4.4 Apresentação do Sistema

O simulador PSOView está disponível para uso em sua versão 1.0 no endereço: http://bsi.ceavi.udesc.br:8080/psoview. Ao acessar o simulador será exibida a tela inicial do sistema. Esta é a única tela, a qual permite utilizar todos os recursos do sistema. A Figura 5 apresenta a tela inicial do simulador.

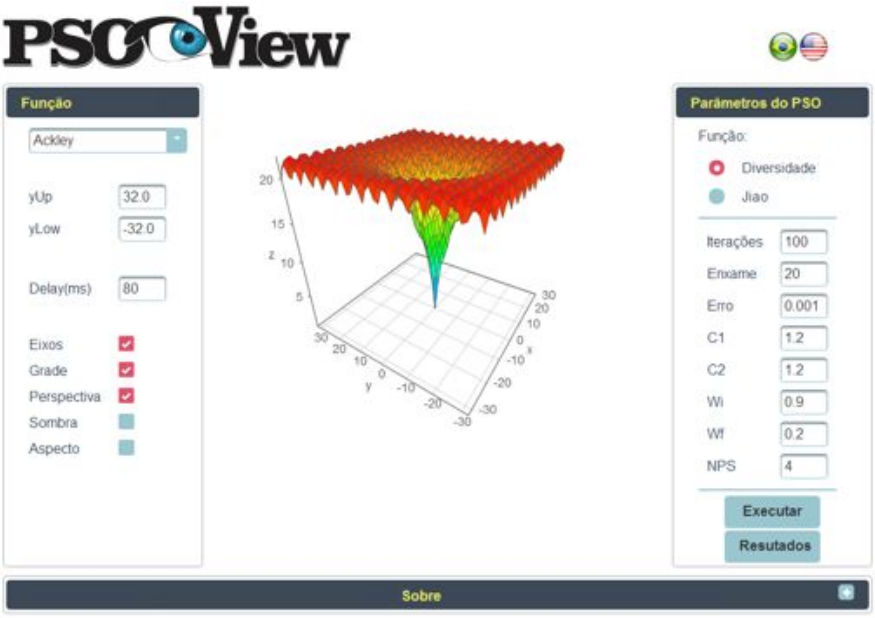


Figura 5. Tela inicial do simulador PSOView

Por padrão, o sistema já apresenta uma função teste ao usuário com configurações predefinidas no idioma português do Brasil. Para alterar a função teste que se deseja minimizar basta clicar sobre a função atual que o sistema apresentará uma lista com todas as funções teste implementadas, inclusive a função definida pelo usuário.

Selecionando uma função qualquer, o sistema irá redesenhar automaticamente o gráfico da função com as configurações predefinidas. O usuário ainda pode construir a sua própria função a partir de uma expressão matemática. Para que isto se torne possível basta selecionar a função: *Definida pelo Usuário*. Ao selecionar esta função aparecerá na tela um campo para inserir a expressão matemática.

Por definição as variáveis que representam os eixos x e y são respectivamente representadas por *x* e *y*. A caixa de texto auxilia o usuário durante a construção da função por meio de um auto completar sugerindo as expressões matemáticas suportadas. A Figura 6 apresenta o campo *Expressão* e suas funcionalidades.

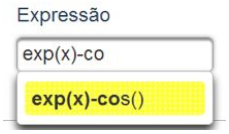


Figura 6. Campo de entrada para a função matemática

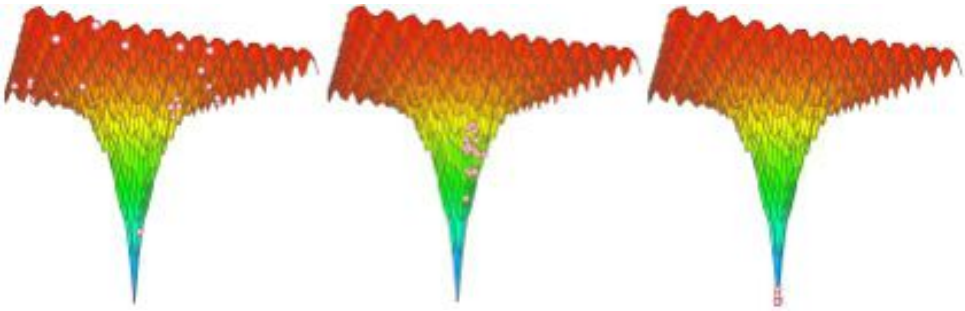
Enquanto o usuário escreve a expressão da função, o simulador automaticamente desenha a função assim que encontrar uma expressão válida. Com a função definida, o usuário pode alterar algumas configurações desta função. Os parâmetros de configuração referem-se ao tamanho dos limitantes e opções de visualização.

Os parâmetros *yUp* e *yLow* referem-se aos limitantes dos eixos x e y da função. Alterando estes parâmetros o simulador automaticamente redesenhará a função com os

novos parâmetros. Já o parâmetro *delay* altera intervalo entre as movimentações do enxame, podendo este ser alterado durante a otimização. Por fim, os parâmetros *Eixos*, *Grade*, *Perspectiva*, *Sombra* e *Aspecto*, não interferem na execução da otimização, apenas na visualização do gráfico da função. Ao clicar em uma opção, o gráfico será redesenhado com a nova configuração, inclusive durante a otimização.

Depois de configurar os parâmetros da função, deve-se ajustar os parâmetros de configuração do algoritmo de otimização. O simulador é bastante flexível, e permite o ajuste de nove parâmetros distintos. Todos os parâmetros já vêm com valores predefinidos, com base no trabalho de Jiao, Lian e Gu (2008). Pode se ajustar os seguintes parâmetros: número máximo de iterações, número de partículas do enxame, coeficientes cognitivo e social, peso inercial inicial e final e critério de parada (erro).

Finalmente, clicando no botão *Executar*, caso os parâmetros informados sejam válidos o sistema inicia o processo de otimização. A cada iteração atualiza-se a posição do enxame na função teste. As partículas do enxame são representadas por esferas. A Figura 7 apresenta o simulador otimizando a função de Ackley em três diferentes estágios. Percebe-se que nas primeiras iterações iniciais as partículas estão dispersas explorando a função, com o decorrer das iterações se aproximam até convergirem para uma região em comum onde está localizado o ponto de mínimo da função.



(a) (b) (c)

Figura 7. PSOView minimizando a função de Ackley. (a) iteração 1, (b) iteração 38, (c) iteração 55

Ao fim da execução a mensagem de término de otimização é exibida e apresenta-se o gráfico da função com a solução encontrada. Para visualizar os demais resultados do processo de otimização, basta clicar no botão *Resultados*. Esta ação fará com que uma caixa de dialogo apareça na tela com os detalhes da solução encontrada. A Figura 8 apresenta a caixa com resultados da otimização.

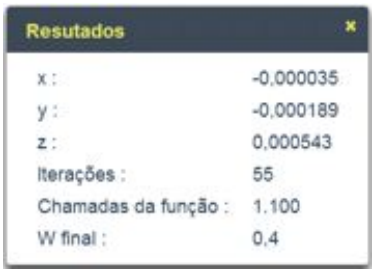


Figura 8. Dialogo de resultados da otimização

4.5 Comparação com Trabalhos Correlatos

Em comparação com os trabalhos correlatos o simulador se mostrou superior na maioria dos recursos comparados, perdendo apenas em alguns detalhes dos quais destacam-se: a execução passo a passo e pausa do processo de otimização. A Tabela 1 apresenta os recursos utilizados na comparação do simulador PSOView com os trabalhos correlatos.

Tabela 1. Comparação entre PSOView e trabalhos correlatos

Recurso	PSOVis	PSO Visualization	PSOView
Criar função teste			X
Alterar limites da função	X		X
Alterar o intervalo de atualização da posição das partículas	X		X
Alterar os parâmetros do PSO	X		X
Cross-Browser	X	X	X
Requer Java 3D instalado	X		
Requer JRE instalado	X	X	
Permitir pausar ou interromper a otimização	X		
Permitir executar passo a passo	X	X	

Como se pode verificar, ao contrário dos demais, o simulador proporciona um ambiente web para a visualização do processo de otimização do algoritmo inteligente PSO, independente de plataforma ou instalação de recursos adicionais. Além de trazer nove famosos problemas de otimização já implementados, o simulador ainda permite a criação de casos de teste customizáveis a partir de expressões matemáticas. Sendo estes os principais diferenciais entre os trabalhos correlatos.

5. Conclusões e Trabalhos Futuros

O simulador se mostrou uma ferramenta eficiente e flexível para visualização do processo de minimização do PSO, podendo servir até mesmo como uma ferramenta didática no ensino de *Swarm Intelligence*. Em geral, o simulador apresenta uma interface gráfica intuitiva e amigável, permitindo a utilização de todos os recursos do simulador em qualquer navegador compatível com HTML5 em uma única tela. Outro grande diferencial é a independência de plataforma ou qualquer outro recurso adicional como JRE ou Java3D. A possibilidade de criar funções teste a partir de expressões matemáticas se mostrou bastante eficiente, pois permite ao usuário criar, visualizar e minimizar seus próprios problemas de otimização de maneira rápida e intuitiva.

Referências

Chap Links Library. (2008) Versão 1.2. Rotterdom, Holand: Almend BUNC. Disponível em: http://chap.almende.com. Acesso em: 05 out. 2012.

Del Valle, Y., Venayagamoorthy, G. K., Mohagheghi, S., Hernandez, J. C. e Harley, R. G. (2008) “Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems”. IEEE Transactions One Evolutionary Computational, v. 12, n. 2. Disponível em: <http://ieeexplore.ieee.org>. Acesso em: 20 jun. 2008.

Direct Web Remoting. (2012) Versão 3.1.rc1. Disponível em: <www.directwebremoting.org/dwr/index.html>. Acesso em: 25 out. 2012.

Graphing Calculator. (2012) Versão 3.0. Ben McCormick, Egor Poblaguev. Disponível em: <www.code.google.com/p/graphingcalculator>. Acesso em: 15 out. 2012.

Harrell, C. R.; Mott, J. R. A., Bateman, R.E., Bowden, R. G. e Gogg, T. J. (2002) "Simulação Otimizando os Sistemas". ed. 1. São Paulo: IMAM e Belge Simulação.

Jiao, B., Lian, Z. e Gu, X. (2008) "A Dynamic Inertia Weight Particle Swarm Optimization Algorithm". Chaos Solitons & Fractals. ed. 37. p.698-705. Disponível em: http://www.sciencedirect.com . Acesso em: 15 jan. 2011.

Kennedy, J., Eberhart, R. C. e Shi. Y. (2001) "Swarm Intelligence". United States of America: Elsevier.

Prime Teknoloji (2012). Prime Faces User Guide. Versão 3.4. Disponível em: < http://primefaces.googlecode.com/files/primefaces_users_guide_3_4.pdf >. Acesso em: 11 jan. 2013.

Pso Visualization. (2004) Versão 1.0. Kent Fritch. Disponível em: <www.projectcomputing.com/resources/psovis>. Acesso em: 16 out. 2012.

Psovis: Modlab. (2005) Versão 1.0. [S.I.]. Disponível em: <www.gecco.org.chemie.uni-frankfurt.de/PsoVis >. Acessado em: 16 out. 2012.

Poli, R., Kennedy, J. e Blackwell, T. (2007) "Particle Swarm Optimization: An overview". Swarm Intelligence. v. 1. p. 33-57.

Rosenbrock, H. H. (1960) "An automatic method for finding the greatest or least value of a function". The Computer Journal, p. 175-184.

Shi Y. e Eberhart R. (1998) "A modified particle swarm optimizer". Proc. IEEE Int. Conf on Evolutionary Computation , p.69-73.

Shi, Y. (2004) "Particle Swarm Optimization". Eletronic Data Systems, Inc. Kokomo, IN 46902, USA, IEEE Neural Networks Society.

Thomas, N. e Reed, M. (2009) A hybrid algorithm for continuous optimization. In *IEEE Congress on Evolutionary Computation* Trondheim. Anais… Trondheim: IEEE 2009, p. 2584-2589.

Van Den Bergh, F. e Engelbrecht, A. P. (2006) "A Study of Particle Swarm Optimization Particle Trajectories". Information sciences ed. 176 p. 937-971. Disponível em: www.elsevier.com/locate/ins>. Acesso em: 15 set. 2011.

Zuben, F. J. V. e Attux, R. R. F. (2008) "Inteligência de Enxame". São Paulo: Unicamp.