UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

Trabalho Prático 2 - Métodos de Pesquisa: Simulated Annealing e Particle Swarm Optimization

---

Engenharia Informática

Inteligência Artificial

José Paulo Barroso de Moura Oliveira

Eduardo José Solteiro Pires

Autores

Diana Ferreira – al68938

Vila Real, 7 de janeiro de 2021

Índice

1. Introdução

Neste trabalho, irão ser expostos dois métodos de pesquisa distintos, o Simulated Annealing e o Particle Swarm Optimization. Os métodos de pesquisa fazem parte da vida diária de todos, pois andamos todos à procura de algo, a pesquisa é feita de forma sistemática. O tipo de métodos de pesquisa que estamos a abordar são associados à inteligência artificial e aos sistemas inteligentes. Existem dois métodos de pesquisa em IA, os métodos não informados (Blind Search), que não utilizam qualquer conhecimento do domínio e os métodos informados, que podem utilizar informação, ou heurística, que ajuda a guiar a pesquisa. A heurística é uma técnica que aumenta a eficiência do processo de pesquisa, sacrificando a exaustão deste. A heurística tem duas possibilidades, ou são muito boas e sugerem direções interessantes para continuar a pesquisa, ou são muito más quando conduzem a direções que levam a becos sem saída. Pesquisa é "o processo de permitir procurar a solução de um problema num conjunto de possibilidades (espaço de estados)." As condições de pesquisa são o estado atual, ou ponto de situação, o objetivo, que é o ponto de situação objetivo, isto é, quando o ponto de situação é igual ao estado objetivo e o custo para obter a solução, que é, em muitos casos, o número de iterações ou movimentos. Um operador define quais as possíveis ações a cada instante e o conjunto de estados para quais é possível a partícula movimentar-se a partir do estado atual chama-se de vizinhança. O espaço de pesquisa é o conjunto de todos os estados atingíveis a partir do estado inicial. Um bom exemplo de um espaço de pesquisa é a resolução de um puzzle, onde o espaço de pesquisa são os movimentos possíveis, o estado objetivo é o puzzle completo e o custo do percurso é o número de movimentos realizados pelo utilizador.

Muitos métodos de pesquisa mais "antigos" na área de inteligência artificial eram realizados a partir de árvores de pesquisa, porém, estas árvores de pesquisa eram muito exaustivas e demoravam demasiado tempo a serem realizadas, pelo que foi necessário o desenvolvimento de novos e melhores métodos de pesquisa. Estes métodos das árvores de pesquisa podem ser apelidados de métodos de "força bruta", visto que são expostas e tidas em consideração todas as possibilidades que o problema poderia tomar. Um exemplo prático onde se verifique que estes métodos não são eficazes, é o cálculo, por exemplo, da iminência de colisão entre dois aviões, se os aviões forem colidir dentro de 3 minutos e estes métodos de pesquisa demorem horas a serem resolvidos, estes não serão considerados muito práticos. A diferença entre os métodos de pesquisa exaustiva e os métodos de pesquisa heurística são que, nos de pesquisa exaustiva, todo o espaço de pesquisa é examinado enquanto que na heurística é utilizado o conhecimento adquirido por

experiência pessoal e, nalguns algoritmos, como o abordado neste trabalho, o PSO, utilizam também variáveis como o conhecimento social para restringir o espaço de pesquisa a um espaço mais pequeno, melhorando assim a eficiência encontrando mais rapidamente o estado objetivo. Os algoritmos genéticos, Simulated Annealing e Particle Swarm Optimization, abordados intensivamente neste relatório, são métodos de pesquisa meta-heurísticos de inspiração natural e biológica, porque são baseados e profundamente influenciados pela observação do comportamento do ser humano e da natureza.

Não se pode falar em métodos de pesquisa sem se falar em otimização. O objetivo da otimização é obter melhorias, alcançar o resultado mais próximo possível do estado objetivo considerado perfeito. Resumindo, um método de pesquisa é classificado com uma função objetivo, um espaço de pesquisa, uma solução para o problema, restrições e podem ser contínuos ou combinatórios, que, respetivamente, significa que possuem um número infinito de soluções viáveis e finito. A evolução de uma pesquisa pode ser considerada determinística, caso apenas haja uma forma de evoluir, onde os problemas podem ser considerados mais simples, e complexos, onde a evolução da pesquisa é considerada estocástica, isto é, a sua evolução irá ser aleatória, sendo difícil prever o que irá acontecer nessa mesma pesquisa. É procedida as análises intensivas dos algoritmos Simulated Annealing e Particle Swarm Optimization.

2. Algoritmos Simulated Annealing e Particle Swarm Optimization

2.1. Introdução aos Algoritmos Simulated Annealing e PSO

2.1.1. Introdução ao Algoritmo Simulated Annealing

O Simulated Annealing A foi desenvolvido em 1983 por Kirkpatrick e foi baseado num algoritmo desenvolvido por Metropolis (1953). A inspiração natural da técnica vem de um processo metalúrgico conhecido como annealing. Neste procedimento metalúrgico os metais são aquecidos a alta temperatura de forma a obter um estado líquido e depois, arrefecidos lentamente para evitar ruturas no material.

O sucesso da aplicação do SA depende do ajustamento correto de uma série de aspetos para cada aplicação, tais como:

- O valor da temperatura inicial;
- A relação de decaimento da temperatura utilizada;
- A lei probabilística utilizada;
- O número de repetições da pesquisa para cada valor da temperatura.

Esta técnica começa a procurar a partir de uma solução inicial qualquer. O procedimento principal consiste em um loop que a cada iteração, gera aleatoriamente um único vizinho 'x' da solução corrente x.

Ao criar um novo vizinho x’ de x, a variação Δ do valor da função é testada, isto é:

(t) = E_new-E(t), em que E_new= x' e E(t)= x (2.1)

Quando esta variação:

- Δ <0: Significa que houve uma redução de energia, o que implica que a nova solução é melhor que a solução anterior. O método aceita a solução e x' passa a ser a nova solução corrente;

- Δ=0: Significa um caso de estabilidade, não havendo nem aumento nem redução de energia. Na realidade, é uma situação improvável de aontecer na prática;

- Δ>0: Significa que houve um aumento de energia. O acontecimento deste tipo de soluções é mais provável a altas temperaturas e bastante improvável a temperaturas reduzidas. Normalmente, para calcular a probabilidade de se aceitar a nova solução,

utiliza-se uma função conhecida por fator de Boltzmann, que é dada por e^(-Δ/T), onde T é um parâmetro do método, chamado de temperatura e que regula a probabilidade de soluções com pior custo. Habitualmente, gera-se um número aleatório entre 0 e 1 e caso o número seja menor ou igual a “p", aceita-se a solução, caso contrário rejeita-se a solução.

Inicialmente, a temperatura, T, assume um valor alto, T0(Temperatura inicial). Após um número fixo de iterações (o qual representa o número de iterações para o sistema atingir o equilíbrio térmico em uma dada temperatura), a temperatura começa a reduzir gradualmente por uma razão de resfriamento α, tal que Tn ← α * Tn -1, sendo 0 < α < 1. O valor da temperatura, ao longo das iterações tende para zero. O algoritmo finaliza quando a temperatura está perto de zero, e não há mais nenhuma solução aceite, ou seja, o sistema está estável. Logo, nessa situação, estaríamos perante um mínimo local.

O algoritmo de Simulated Annealing desenvolvido no contexto deste projeto apresenta-se seguidamente:

```
2 -  quality = @(x1,x2) -20*exp(-0.2*sqrt(1/2*(x1.^2+x2.^2)))-exp(1/2*(cos(x1)+cos(x2)))+ 20 + exp(1); %Função fornecida
3
4    %Estabelecimento dos limites e design do gráfico%%%
5 -  x1 = linspace(-32.768,32.768,100);                %
6 -  x2 = linspace(-32.768,32.768,100);                %
7 -  [X1,X2] =  meshgrid(x1,x2);                       %
8 -  Fx = quality(X1,X2);                              %
9 -  contour(X1,X2,Fx,20);                             %
10   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11
12 -  T = 90; %Inicialização da variável temperatura (para proceder-mos a uma analogia entre a temperatura de um metal
13           %e este parâmetro ajustável do algoritmo, visto que este se baseia numa analogia deste tipo
14
15 -  r = (rand(1,2)-0.5)*2*32.768; %Gera uma matriz de 1 por 2 de números aleatórios entre -32.768 e 32.768
16
17 -  fx = quality(r(1), r(2)); %Cálculo do valor da qualidade da função segundo os valores gerados aleatóriamente
18
19 -  hold on %Retém o gráfico atual para que novos plots sejam realizados no mesmo gráfico do inicial
20
21 -  plot(r(1), r(2), 'r*') %Coloca a partícula no gráfico
22 -  xlabel('x') %Label do eixo dos x
23 -  ylabel('y') %Label do eixo dos y
24 -  title('Simulated Annealing') %Título do gráfico
25
```

Figura 2.1.1.1 – Definição inicial das variáveis e função

Na figura 2.1.1.1, é definida a função de Ackley, função que foi considerada para a realização do projeto (linha 2). Seguidamente, foi realizada a definição dos limites do gráfico. Foi considerada a temperatura inicial (T) de 90 no âmbito deste algoritmo. A próxima fase é a geração de uma matriz de 1 linha por 2 colunas, isto é, uma matriz com duas entradas, para

armazenamento das coordenadas aleatórias do que será a primeira partícula com a equação seguinte:

$$r = (rand(1,2) – 0.5) * 2 * 32.768; \quad (2.2)$$

Nesta equação, a variável ‘r’ é criada e nela são armazenadas 2 variáveis aleatórias, mas, no limite estabelecido de [-32.768 ; 33.768], pois são estes os limites do gráfico.

Na linha 17 da figura 2.1.1.1, é calculada a qualidade, isto é, o fx, das coordenadas geradas aleatoriamente previamente, relativamente ao gráfico da função estipulada.

Seguidamente nas linhas 19-24 a partícula é inserida no gráfico da função e é-lhe atribuído os nomes dos seus necessários atributos, como o título e os nomes das variáveis.

No seguinte segmento de código de algoritmo:

```
26 -    n = 0; %t e n são variáveis inicializadas para que possamos estabelecer o número de iterações do ciclo
27 -    t = 1;
28
29 -    evo(t) = 0; %Inicialização da variável evo para verificarmos a evolução da qualidade da função, isto é, do fx
30 -    temp(t) = 0; %Inicialização da variável temp para verificarmos a evolução da temepratura, isto é, de T
31 -    probabilidade(t) = 0; %Inicialização da variável para verificarmos a evolução da probabilidade, isto é, do prob
32
33 -    while(t <= 1000) %Inicialização do ciclo while para que o algoritmo funcione
34
35 -        n = 0; %Reinicialização da variável n para que o ciclo while seguinte seja percorrido 5 vezes por cada
36              %vez que o ciclo while em que este se insere percorre
37
38 -        while(n <= 5) %Inicilização de um segundo ciclo while
39
40 -            r_new = r + (rand(1,2)-0.5)*2*1; %Geração de novas variáveis aleatórias na vizinhança das precedentes (r)
41
42 -            r_new = max(min(r_new,32.768),-32.768); %Estabelecimento de um limite na geração das novas variáveis
43                                                    %para que estas não ultrapassem os limites do gráfico e não saiam
44                                                    %do domínio deste
45
46 -            fx_new = quality(r_new(1), r_new(2)); %Cálculo da qualidade da nova partícula que irá ser potencialmente gerada
47
48 -            deltaE = fx_new - fx; %Cálculo da variância da energia ao longo do ciclo
49
50 -            prob = exp(-deltaE/T); %Cálculo da probabilidade de aceitação da nova partícula, que irá ser menor conforme
51                                    %o número de iterações
52
53 -            prob = max(min(prob,1),0); %Para que a probabilidade não ultrapasse 1, pois isso é matematicamente impossível
```

Figura 2.1.1.2 – Inicialização dos ciclos while e cálculo do ΔE e da probabilidade de aceitação de variáveis piores que as previamente aceites

Neste segmento, são inicializados os ciclos while, para dar início à pesquisa do resultado ótimo. Com isto, são também geradas novas matrizes, ‘evo(t)’, para guardar a evolução do ‘fx’ ao longo das iterações, ‘temp(t)’, para armazenar a evolução da temperatura ao longo das iterações e a variável ‘probabilidade(t)’ para guardar a evolução da probabilidade de aceitação.

Dentro do ciclo while (neste caso foram estipuladas 1000 iterações como o número de iterações que irão ser realizadas para testar a eficácia do algoritmo) é gerada uma nova coordenada na vizinhança da partícula colocada aleatoriamente no espaço de pesquisa com a função:

r_new = r + (rand(1,2)-0.5) *2 *1; (2.3)

Na função 2.3, é incrementada à coordenada da partícula já existente um valor entre -1 e 1, para que a nova partícula gerada seja colocada no espaço de pesquisa muito próximo da partícula anterior. Com a função 'max' e 'min' do Matlab é estipulado limite do espaço de pesquisa, para que as partículas não saiam do espaço.

Na linha 46 é calculada a qualidade (fx) da nova partícula, para, futuramente, poder ser comparada com a partícula anterior. Seguidamente, é calculado o ΔE, com a seguinte equação:

$$\Delta E = fx\_new - fx \quad (2.4)$$

ou seja, está a ser calculada a variação da energia para que, seguidamente, possa ser decidido se a nova partícula reúne os critérios necessários à adição ao espaço de pesquisa ou não. Como já foi visto anteriormente, caso ΔE seja menor que 0, a solução será aceite, caso contrário irá ter de ser tido em conta um novo critério, o critério da probabilidade de ser aceite uma pior solução. Para isso, é calculada a probabilidade através da seguinte fórmula:

$$prob = e^{-\Delta E/T} \quad (2.5)$$

com o valor max e min de prob estabelece-se que a probabilidade, matematicamente, tem que estar entre os valores $0 < prob < 1$.

```
55 -        if deltaE < 0 %Caso a amplitude das energias seja menor que 0, a nova partícula é automaticamente aceite e a
56                       %anterior substituída pela nova
57
58 -             r = r_new; %As coordenadas aleatórias geradas anteriormente são substituídas pelas da nova partícula
59 -             fx = fx_new; %A qualidade, ou fx, da partícula anterior é substituída pela da nova partícula
60 -             plot(r(1), r(2), 'gx') %A nova partícula é colocada no gráfico
61
62 -        elseif rand < prob %Caso a condição anterior não seja verificada, é feita uma nova verificação e garantida uma
63                            %segunda oportunidade para que a partícula nova possa ser aceite. Se a probabilidade,
64                            %calculada anteriormente, for maior que um número gerado  aleatóriamente entre 0 e 1,
65                            %esta nova partícula, com fx maior que aanterior, o que não é o inicialmente pretendido,
66                            %é aceite e substitui a anterior
67
68 -             r = r_new; %As coordenadas aleatórias geradas anteriormente são substituídas pelas da nova partícula
69 -             fx = fx_new; %A qualidade, ou fx, da partícula anterior é substituída pela da nova partícula
70 -             plot(r(1), r(2), 'yo') %A nova partícula é colocada no gráfico
71
72 -        end %Fim da constraint if
73
74 -             evo(t) = fx; %No final de cada ciclo, na matrix evo é registada a qualidade da partícula
75 -             n = n + 1; %A cada ciclo n incrementa 1 valor para que o ciclo while funcione corretamente
76
```

Figura 2.1.1.3 – Atualização das variáveis

Na figura anterior, verifica-se a realização das comparações, com utilização das constraints if.

Caso ΔE seja menor que 0, a nova partícula é automaticamente aceite e a anterior substituída pela nova, ou seja, as coordenadas da partícula antiga são substituídas pelas coordenadas da nova partícula assim como o seu fx. Seguidamente, para motivos de comparação e verificação da evolução das partículas, a nova partícula é colocada no espaço de pesquisa para futura referência.

Caso ΔE seja maior do que 0, ainda há uma probabilidade, que irá decrescer ao longo das iterações, devido ao decaimento da temperatura (T), que irá ser abordado mais tarde, da nova partícula ser aceite, é gerado um número aleatório entre 0 e 1, caso a probabilidade anteriormente calculada seja maior do que este número gerado aleatoriamente, a nova partícula, com pior solução que a gerada na anterior iteração, irá ser aceite de qualquer das formas e substituirá as coordenadas e valor de fx da partícula anterior, bem como será inserida também no espaço de pesquisa.

Com as linhas de código 74 e 75, armazena-se o valor de fx da iteração atual para futura referência e, ao n, é incrementado 1 valor, para que o ciclo while seja percorrido de 1 em 1 e o ciclo while interno termina.

Dentro ainda do ciclo while externo, faltam ser analisados alguns pontos:

```
83 -        probabilidade(t) = prob; % Registo da evolução da probabilidade a cada ciclo, para ser analisada
84
85 -        T = 0.94*T; %Decaimento da temperatura ao longo do ciclo while, com um fator de 0,94
86
87 -        t = t + 1; %A cada ciclo t incrementa 1 valor para que o ciclo while funcione corretamente
88
89 -    end %Fim do ciclo while externo
90
91 -    plot(r(1),r(2),'b*') %Partícula melhor, ou seja, com o menor valor de fx econtrada é colocada no gráfico
92                          %com uma cor diferente das outras, para que possa ser facilmente identificada como a melhor
93
94 -    hold off %Liberta o gráfico que estava atualmente retido
```

Figura 2.1.1.4 – Decaimento da temperatura e fim do ciclo da pesquisa

Fora do ciclo interno while mas ainda dentro do ciclo de pesquisa, as probabilidades a cada iteração são armazenadas dentro da matriz ‘probabilidade(t)’ e procede-se ao cálculo do decaimento da temperatura, considerando, no âmbito deste projeto, com um fator de 0,94, através do uso da seguinte fórmula (linha 85):

T = 0.94 * T: (2.5)

Com o decair da temperatura, a probabilidade de aceitação de soluções piores que a prévia vai reduzindo conforme o número de iterações até que atinja o valor de 0, onde o ciclo de pesquisa já não vai mais aceitar soluções consideradas piores que a anterior como possíveis soluções para o espaço de pesquisa.

Com isto, está encerrado o ciclo de pesquisa e passa-se à inserção da melhor posição encontrada no gráfico, com o código da linha 91. Para finalizar a análise deste algoritmo, ainda é considerada a seguinte porção de código:

```
96 -    figure %Figure cria uma nova janela com uma nova figura, no nosso caso um gráfico para que possamos inserir os dados
97 -    plot(evo) %Insere os dados da evolução do fx
98 -    ylabel('Fx') %Label do eixo dos y
99 -    xlabel('Iterações') %Label do eixo dos x-
100 -   title('Evolução do fx') %Título do gráfico
101
102 -   figure
103 -   plot(temp) %Insere os dados da evolução da temperatura
104 -   ylabel('T') %Label do eixo dos y
105 -   xlabel('Iterações') %Label do eixo dos x
106 -   title('Evolução da temperatura') %Título do gráfico
107
108 -   figure
109 -   plot(probabilidade) %Insere os dados da evolução da probabilidade
110 -   ylabel('Probabilidade') %Label do eixo dos y
111 -   xlabel('Iterações') %Label do eixo dos x
112 -   title('Evolução da probabilidade') %Título do gráfico
```

Figura 2.1.1.5 – Criação dos gráficos de evolução das variáveis

Com estas linhas de código são inseridas num gráfico as evoluções das variáveis consideradas no algoritmo conforme o número de iterações, para poderem ser comparadas futuramente (ver página x, capítulo 2.2.1).

2.1.2. Introdução ao Algoritmo Particle Swarm Optimization (PSO)

O algoritmo do Particle Swarm Optimization (PSO), é baseado na observação do comportamento dos animais na natureza, particularmente como enxames. Este tipo de comportamento despertou sempre muita curiosidade por parte dos investigadores:

- Há vantagens neste tipo de comportamento?
- Há desvantagens?

Estas foram perguntas que os investigadores sempre se foram colocando e, após observação intensiva destes tipos de comportamentos, chegou-se à conclusão de que as vantagens da partilha social da informação superavam claramente as desvantagens da competição. As presas tendem mais a movimentarem-se em grupos do que os predadores. Por exemplo, num rebanho de ovelhas, estas juntam-se para que as do centro fiquem protegidas de predadores, caso não houvesse este comportamento e estas se isolassem umas das outras, seriam claramente uma causa perdida uma ovelha solitária enfrentar, por exemplo, um lobo, visto que este possui inúmeras vantagens sob a primeira. Para isto, as ovelhas tiveram de se adaptar e trabalhar com o que tinham, neste caso, elevado número de indivíduos, daqui formaram-se os rebanhos ou enxames, trabalhar em grupo para manter a sobrevivência. Um enxame é definido como um conjunto de agentes que se movem uniformemente capazes de comunicarem uns com os outros direta ou indiretamente, resolvendo coletivamente um problema. Baseando-se nesta ideologia, surgiu uma aplicação computacional pioneira, o BOIDS, desenvolvida por Craig Reynolds, profundamente inspirado por esta mentalidade dos animais de “enxame”. O objetivo do BOIDS é simular graficamente, através de animações computarizadas, o movimento coordenado de um bando de pássaros. Este propôs três regras fundamentais para simular computacionalmente este comportamento:

- Coesão (C) – Promove o agrupamento dos Boids em torno da média das suas posições;
- Separação (S) – Previne o choque dos Boids, evitando que cada Boid colida e invada o espaço de outro Boid, havendo distanciamento social;
- Alinhamento (A) – Promove o alinhamento da velocidade de um Boid com a dos seus vizinhos;

A posição de cada Boid, i, pode ser determinada numa iteração t+1, através da anterior, t, através da seguinte expressão:

$$x(t+1) = x(t) + v(t+1) \quad (2.6)$$

Já a velocidade utilizada na anterior expressão (2.6), é determinada pela expressão:

$$V(t+1) = v(t) + \beta_1 vc(t) + \beta_2 vs(t) + \beta_3 va(t) \quad (2.7)$$

O PSO segue um pensamento semelhante. Proposto originalmente por Kennedy e Eberhart, foi colocada a seguinte questão: “Como é que cada agente decide para onde se vai mover no espaço?” O deslocamento de cada agente é efetuado baseando-se na sua própria experiência bem como na experiência de grupo. Cada partícula i, é caracterizada por duas variáveis, a sua velocidade, v, e a sua posição, x. A cada iteração, é guardada a melhor posição atingida por cada partícula, b, e a melhor posição global, g, isto é, a melhor solução para o problema dentro da matriz das melhores posições de cada uma das partículas.

As duas equações que constituem o fundamento base do cálculo das soluções do PSO são representadas seguidamente:

- Atualização da velocidade de cada partícula, a cada iteração:

$$v_i(t+1) = \omega v_i(t) + c_1\varphi_1(b_i(t)-x_i(t)) + c_2\varphi_2(g(t)-x_i(t)) \quad (2.8)$$

- Atualização da posição de cada partícula, a cada iteração:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2.9)$$

Analisando a primeira equação, podemos identificar várias variáveis:

- $v_i(t+1)$ – velocidade atualizada das partículas
- $v_i(t)$ – velocidade da iteração anterior
- c_1 – constante cognitiva
- φ_1 – número aleatório que representa o fator cognitivo e irá ser influenciado pela constante cognitiva
- $b_i(t)$ – melhores posições de cada partícula
- $x_i(t)$ – posição atual das partículas
- c_2 – constante social
- φ_2 - número aleatório que representa o fator social e irá ser influenciado pela constante social
- $g(t)$ – melhor posição global
- ω - fator de inércia

Com estas informações, pode-se verificar que, caso não houvesse componente social, apenas a componente cognitiva afetaria o movimento das partículas, e já não seria considerado um algoritmo baseado em enxames, por isso, esta porção da equação é muito importante para este tema.

Mais tarde, foi acrescentado à equação 2.8, o fator de inércia, ω, que melhora significativamente o desempenho do PSO. Esta variável, normalmente, decai linearmente ao

longo das iterações, estabelecendo um compromisso importante entre exploração e especialização.

Segue-se uma explicação breve do código desenvolvido no âmbito deste projeto para sucessivos testes e comparação com o algoritmo do Simulated Annealing:

```
14 -    x = (rand(5,2)-0.5)*2*32.768; %Matriz de coordenadas aleatórias
15
16 -    fx = quality(x(:,1), x(:,2)); %Quality inicial de cada partícula
17
18 -    plot(x(1,1), x(1,2), 'rx') %Colocar as 5 particulas inicializadas aleatoriamente num gráfico
19 -    plot(x(2,1), x(2,2), 'bx')
20 -    plot(x(3,1), x(3,2), 'gx')
21 -    plot(x(4,1), x(4,2), 'kx')
22 -    plot(x(5,1), x(5,2), 'cx')
23
24 -    xlabel('Iterações'); %Eixo dos x e dos y assim como o título do gráfico são definidos aqui
25 -    ylabel('fx');
26 -    title('Particle Swarm Optimization');
27
28 -    b = x; %Inicialização da matriz de coordenadas com os personal best de cada partícula
29 -    fb = fx; %Inicialização da matriz das qualidades personal best de cada partícula
30
31 -    [fg, i] = min(fb); %Inicialização da melhor quality global, isto é, o min encontra o valor mínimo da matriz
32                         %com os melhores fx pessoais de cada partícula, o melhor é atribuído ao valor fg
33 -    g = b(i,:); %Inicialização das coordenadas global best
34
35 -    v = (rand(5,2)-0.5)*2*1; %Velocidade inicial de cada particula gerada aleatoriamente entre 0 e 1
36
37 -    n = 1; %Inicializador do loop while
38 -    w = 0.7; %Fator de inércia
```

Figura 2.1.2.1 – Inicialização das partículas

Nesta parte inicial do código do algoritmo do PSO, é gerada uma matriz de 5 linhas por 2 colunas, isto é, são geradas 5 partículas aleatórias com as suas devidas coordenadas (2), coordenadas estas com os seus valores restringidos ao espaço de pesquisa, neste caso, -[32.768 ; 32.768]. É calculada a qualidade, fx, de cada uma das partículas, com a linha 16, e são colocadas as partículas no espaço de pesquisa (linhas 18-22). Inicializam-se as matrizes de variáveis com as coordenadas e fx de cada uma das partículas, b e fb, respetivamente, e a matriz com as coordenadas da melhor posição global, b assim como o fx dessa melhor posição, g. A velocidade inicial de cada partícula é também estabelecida (linha 35) e, para este caso, foi considerada uma velocidade inicial aleatória dentro do intervalo [-1, 1]. O fator de inércia, ω, foi também inicializado com o valor 0.7. Na figura seguinte, é inicializado o ciclo de pesquisa e calculada a velocidade:

```
40 -    bestred(n) = 0;     %Inicialização das variáveis para armazenar os personal e globalbests de
41 -    bestblue(n) = 0;    %cada partícula para futura referência
42 -    bestgreen(n) = 0;
43 -    bestblack(n) = 0;
44 -    bestcyan(n) = 0;
45 -    globalbest(n) = 0;
46
47 -  while n <= 1000 %Inicialização do ciclo while, calculado para que w seja 0.4 no final do ciclo
48
49 -      phic = (rand(5,2)-0.5)*2*1; %Inicialização do phi cognitivo, erro cognitivo, utilizado na equação de
50                                   %evolução de velocidade da partícula
51 -      phis = (rand(5,2)-0.5)*2*1; %Inicialização do phi sociial, erro social, utilizado na equação de
52                                   %evolução de velocidade da partícula
53
54 -      w = w - 0.0003; %Evolução do fator de inércia ao longo do ciclo
55
56 -      v = w * v + 0.2 * phic.*(b - x) + 0.2 * phis.*(g - x); %Equação de determinação de velocidade das partículas
57                                                              %ao longo do ciclo,onde a velocidade aumenta com cada
58       %Nota: foram considerados os valores 1.5 para         %loop onde w representa o fator de inércia, v a
59       %      os valores de c1 e c2 porque, devido ao        %velocidade anterior, 1.5 é o c1 ou seja, a
60       %      baixo número de iterações, com o valor         %componente de peso para o fator de erro cognitivo,
61       %      2 iriam raramente todas as partículas          %phic é o erro cognitivo, (b-x) significa o valor
62       %      converger para o mesmo valor                   %das coordenadas melhores de cada partícula menos o
63                                                              %valor gerado aleatoriamente, o segundo 1.5
64                                                              %representa o c2, ou seja um peso para o fator social,
65                                                              %phis e (g-x) são as coordenadas melhores
66                                                              %globais menos as coordenadas geradas aleatoriamente
```

Figura 2.1.2.2 – Ciclo e cálculo da inércia e velocidade

Com as linhas 40-45, são inicializados os valores das matrizes que serão criadas para armazenar os melhores valores de cada partícula a cada iteração assim como o global best, para mais tarde ser referenciado.

Dentro do ciclo de pesquisa, são então inicializados os φ cognitivo (phic, linha 49) e o φ social (phis, linha 51). Neste caso, são lhes atribuídos valores aleatórios do intervalo [-1 ; 1].

Na linha 54, é também evidenciada a equação de decaimento do fator de inércia, ω, que foi assim estabelecida para que, com o número de iterações, 1000, esta acabasse com o valor de 0,4, tendo assim em consideração um decaimento linear entre 0.7 e 0.4 com a seguinte equação:

$$\omega = \omega - 0.0003 \quad (2.10)$$

Na linha 56, é evidenciada uma das duas equações que constituem o coração do algoritmo PSO, a equação de atualização da velocidade. Como já foi visto e analisado previamente (ver página 10, equação 2.8), esta equação é constituída por várias variáveis diferentes, dividindo a equação em duas partes distintas: a parte cognitiva, que representa o

pensamento pessoal de cada partícula, não sendo este afetado pelo enxame: $c_1\varphi_1(b_i(t)-x_i(t))$ e a parte social, que introduz à partícula o pensamento social do enxame, que irá influenciar o modo de movimentação da mesma, influenciando os passos seguintes da partícula, isto é, para onde esta se irá movimentar: $c_2\varphi_2(g(t)-x_i(t)$. Esta equação é muito importante para o passo seguinte, onde é evidenciada a atualização da posição da partícula no espaço de pesquisa:

```
68 -      x = x + v; %Equação de atualização das coordenadas de cada partícula, onde x representa a matriz das
69                   %coordenadas de cada uma das partículas e v a velocidade determinada na equação anterior
70
71    %Estas duas equações representam o coração do PSO, isto é, a partir destas duas equações é que o PSO funciona corretamente
72
73 -      x = max(min(x,32.768),-32.768); %Estabelecimento de um limite no cálculo das novas coordenadas para que estas não
74                                        %ultrapassem os limites do gráfico e não saiam do domínio deste
75
76 -      fx = quality(x(:,1), x(:,2)); %Determinação do novo fx de cada partícula, ou seja, da qualidade nova de cada partícula
77
78 -      for i = 1:1:5 %Ciclo for para determinar se o valor atual verifica um valor menor do que o previamente estipulado
79 -      if fx(i) < fb(i) %Caso isto seja verdade:
80 -          fb(i) = fx(i); %O valor anterior no local (i, 1) da matriz será substituído pela nova coordenada
81 -          b(i,1) = x(i,1); %A coordenada anterior no local (i, i) da matriz será substituída pela nova melhor coordenada
82 -          b(i,2) = x(i,2);
83 -      end %Fim do ciclo if
84 -      end %Fim do ciclo for
85
86 -      [fg, i] = min(fb); %Determinação do melhor valor (quality, fx) global
87 -      g = b(i,:); %Determinação da melhor coordenada global
88
89 -      plot(b(1,1), b(1,2), 'ro') %Inserção no gráfico de cada uma das partículas conforme encontram melhores pessoais
90 -      plot(b(2,1), b(2,2), 'bo')
91 -      plot(b(3,1), b(3,2), 'go')
92 -      plot(b(4,1), b(4,2), 'ko')
93 -      plot(b(5,1), b(5,2), 'co')
```

Figura 2.1.2.3 – Atualização da posição das partículas e comparação com as anteriores

Neste segmento, ocorrem várias etapas importantes do algoritmo. A primeira que podemos evidenciar é, tal como foi dito anteriormente, a atualização da posição de cada partícula no espaço de pesquisa através da equação vista anteriormente (ver página 10, equação 2.9), onde, a cada iteração, à posição anterior é acrescentada o valor da velocidade calculada no passo anterior. São definidos os limites de x, [-32.768 ; 32.768] para que a partícula não saia do espaço de pesquisa. Seguidamente é calculada a qualidade, fx, das novas partículas às quais foram incrementadas a velocidade.

Com isto, segue-se um ciclo for (linha 78-84) para percorrer a matriz das qualidades , fx, atuais e compará-las com as qualidades melhores pessoais, fb. Caso as posições atuais sejam melhores que as consideradas anteriormente melhores pessoais, as melhores pessoais são substituídas pelas novas, assim como a sua coordenada. Procede-se, no final de cada ciclo, a uma nova análise das matrizes de melhores pessoais e, caso haja algum valor melhor que o

anterior, o valor anterior é substituído pelo novo melhor global (ver linhas 86-87 da Figura 2.1.2.3)

Nas linhas 89-93 da figura são inseridas todas as 5 partículas, com cores diferentes para que sejam facilmente distinguíveis, conforme os novos valores considerados melhores pessoais. Com tudo isto feito, falta apenas um passo final:

```
100 -       globalbest(n) = fg;
101 -
102 -       n = n + 1; %A cada ciclo n incrementa 1 valor para que o ciclo while funcione corretamente
103 -
104 -   end %Fim do ciclo while
105 -
106 -   plot(g(1),g(2), 'k*'); %Plot da melhor variável global encontrada, ou seja, o valor mínimo encontrado
107 -
108 -   hold off %Liberta o gráfico que estava atualmente retido
109 -
110 -   figure %Gráfico à parte para visualizar-mos a evolução do melhor fx global ao longo das iterações, o fg
111 -   plot(globalbest);
112 -   xlabel('Iterações');
113 -   ylabel('fx');
114 -   title('Evolução do melhor fx global');
115 -
116 -   figure %Gráfico à parte para visualizarmos a evolução dos melhores fx pessoais ao longo das iterações, os fb
117 -   hold on
118 -   plot(bestred, 'r')
119 -   plot(bestblue, 'b')
120 -   plot(bestgreen, 'g')
121 -   plot(bestblack, 'k')
122 -   plot(bestcyan, 'c')
123 -   xlabel('Iterações');
124 -   ylabel('fx');
125 -   title('Evolução dos personal best por partícula');
126 -   hold off
```

Figura 2.1.2.4 – Colocação do global best no espaço de pesquisa e criação dos gráficos para comparação dos dados

Nesta etapa, o melhor global a cada iteração é armazenado numa matriz, ‘globalbest(n)’, para ser referenciada futuramente e o ciclo de pesquisa acaba.

Para terminar, são criados os gráficos com as evoluções das qualidades, fb, de cada uma das partículas assim como a evolução da posição do melhor global.

2.2. Resolução do Problema com os algoritmos PSO e SA

No contexto deste projeto, foi fornecida a equação de Ackley:

$$f(\vec{x}) = -a\,exp(-b\,(\sqrt{\tfrac{1}{n}\sum_{ni=1} xi2}\,)) - exp\,(\tfrac{1}{n}\sum_{ni=1} cos(cx_i)\,) + a + exp(\,1) \quad (2.11)$$

Com as seguintes restrições: $a = 20$, $b = 0.2$, $c = 2\text{pi}$ e $x_i \in [-32.768, 32.768]$

E foi pedido que fosse encontrado mínimo da função. Sabe-se que o mínimo global da função de Ackley se encontra no ponto (0, 0) e tem o valor de 0.

Com os resultados obtidos aplicando o uso dos algoritmos do Simulated Annealing (ver página 3, capítulo 2.1.1) e do Particle Swarm Optimization (ver página 9, capítulo 2.1.2), procede-se a uma análise e posterior comparação dos resultados obtidos com o uso de cada um destes algoritmos.

2.2.1. Resolução do Problema com o Algoritmo Simulated Annealing

Para finalidades de exposição de resultados, foram feitos 6 testes com o algoritmo do Simulated Annealing para tentar encontrar o mínimo global da função 2.11, no espaço de pesquisa estipulado anteriormente ($x_i \in [-32.768, 32.768]$). Passa-se então à exposição dos resultados obtidos:

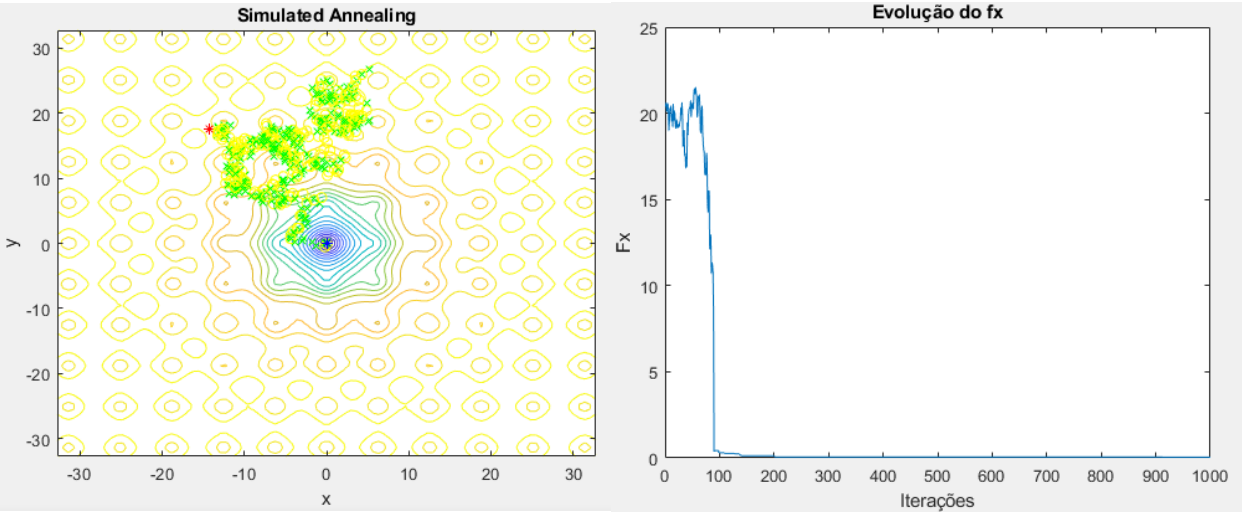

Figura 2.2.1.1 – Teste 1

Neste primeiro teste, pode-se verificar através da análise da figura 1, no gráfico “Simulated Annealing” que, o mínimo global foi encontrado, ao fim de cerca de 140 iterações, como se pode verificar no gráfico “Evolução do fx”. Este foi um resultado considerado ótimo pois foi perfeito e o mínimo encontrado.

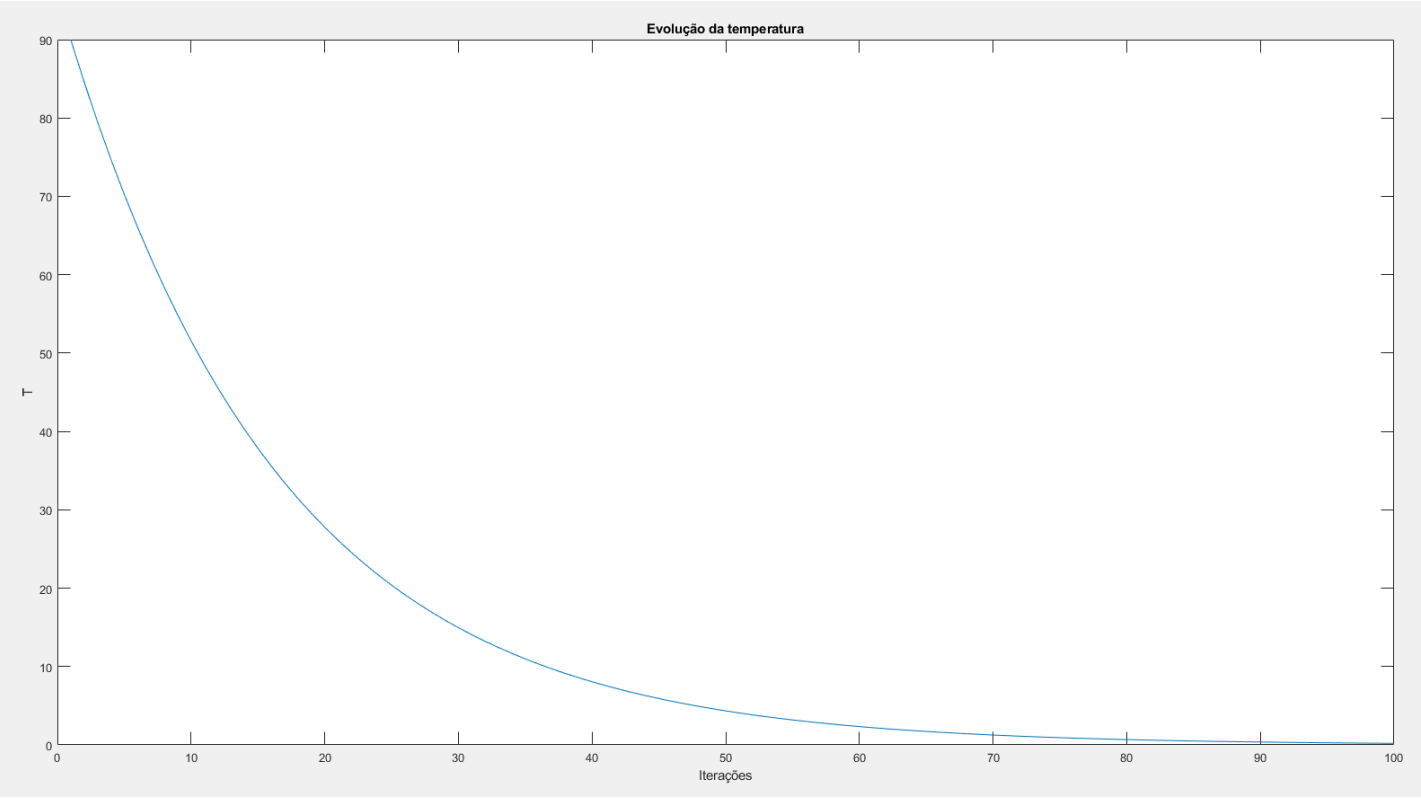

Figura 2.2.1.2 – Evolução da Temperatura

Pode ser também observada a evolução da temperatura, que não irá ser abordado novamente porque será sempre igual, visto que a temperatura inicial e ritmo de decaimento da temperatura não foram alterados entre testes. Com uma temperatura (T) inicial de 90 e um fator de decaimento de 0,94, verifica-se que, ao fim de 100 iterações, a temperatura alcança o valor mínimo possível de 0. O que acontece quando isto se verifica é que a probabilidade de aceitação de novos resultados considerados piores que o anterior é de 0, ou seja, só novas soluções consideradas melhores é que vão ser aceites.

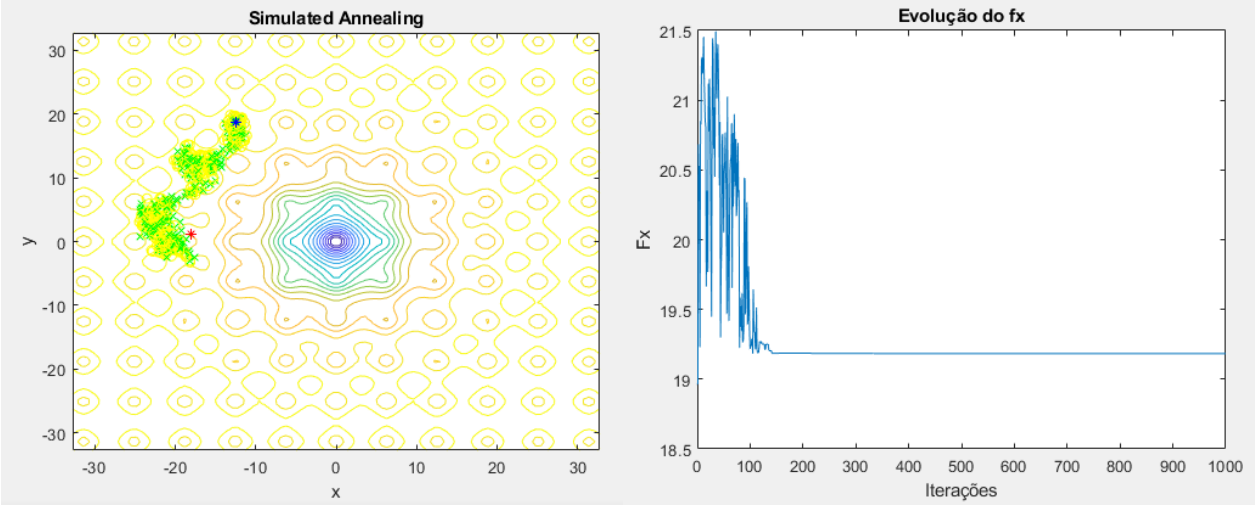

Figura 2.2.1.3 – Teste 2

No segundo teste realizado, o resultado obtido já não foi ótimo, nem perto disso, verificamos assim uma falha neste algoritmo, onde a partícula encontrou um beco sem saída, isto aconteceu ao final de cerca de 120 iterações porque a partícula “caiu num buraco” e não conseguiu de lá mais sair porque a sua probabilidade de voltar a subir do buraco, isto é, de encontrar uma solução pior (mais alta, porque estamos a efetuar uma minimização da função), era de 0, visto que, a partir da iteração 100, como a temperatura é 0, a probabilidade de aceitação de novas soluções piores é de 0. Com isto, a partícula encontrou o mínimo local (-12,5129; 18,7795) que equivale a fx = 19.1830, que é diferente, como já sabemos, do mínmo global, 0. Este não foi um resultado ótimo e prova que o Simulated Annealing não resolve todos os problemas de otimização, havendo espaço para melhorias.

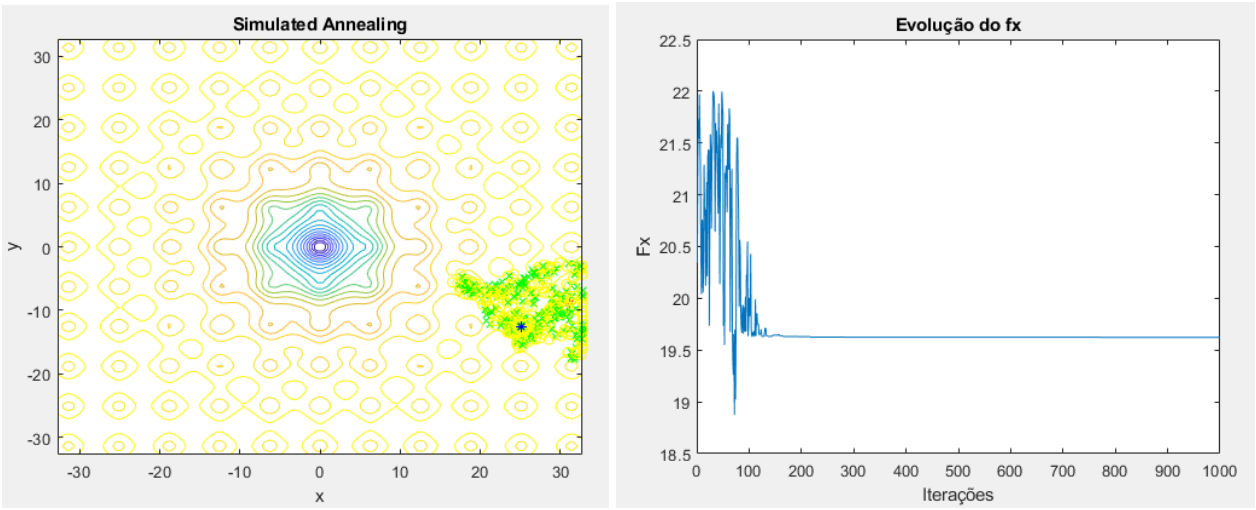

Figura 2.2.1.4 – Teste 3

Com a realização do teste 3, verifica-se que o resultado extraviado obtido no Teste 2 (Figura 2.2.1.3) não foi uma exceção, havendo uma taxa bastante alta de insucessos na performance deste mesmo algoritmo, tendo este ficado “empancado” nas coordenadas (25,0992; -12,5394) que correspondem ao valor fx = 19,6230, que, como já foi previamente verificado, não é o mínimo global da função.

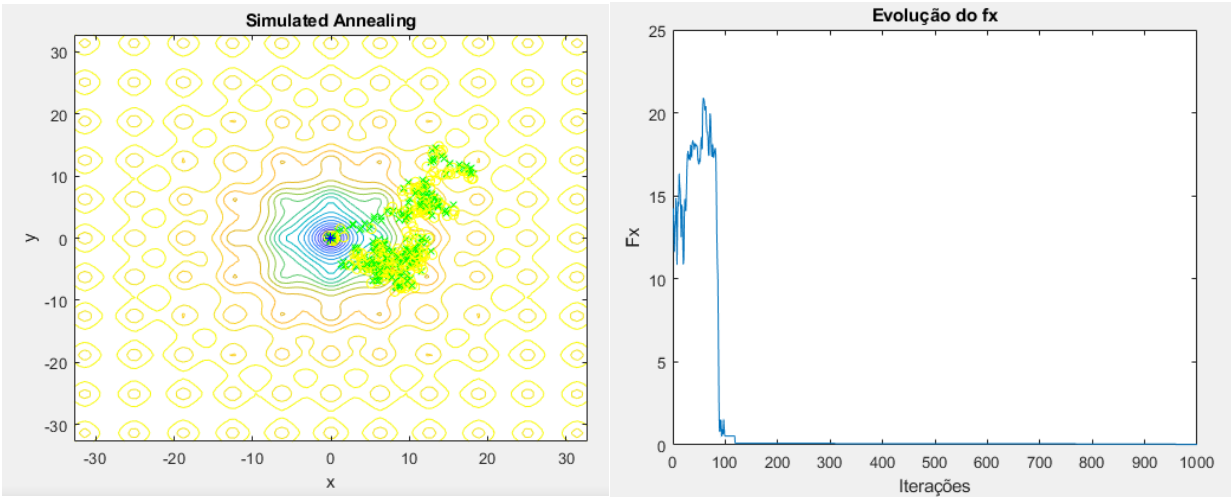

Figura 2.2.1.5 – Teste 4

No Teste 4, verifica-se um retorno aos testes realizados com sucesso, tendo levado cerca de 110 iterações para atingir o resultado quase considerado ótimo de fx = 0.0270, de coordenadas (-0,0095; -0,000099875). Evidencia-se aqui um resultado que pode ser considerado ótimo, ou quase ótimo, visto que este foi extremamente próximo do resultado perfeito.

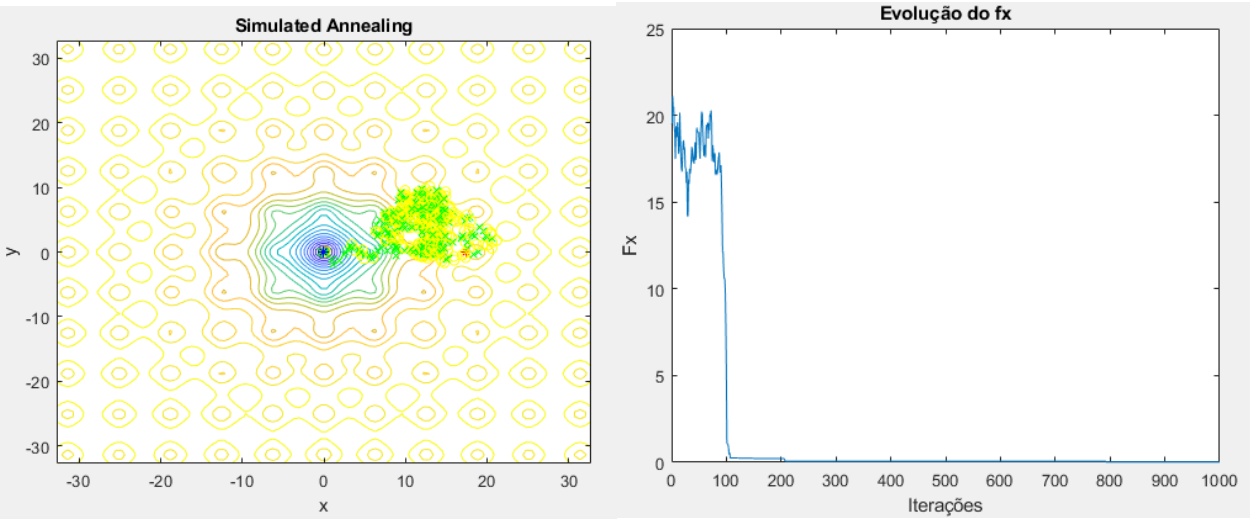

Figura 2.2.1.6 – Teste 5

Com o teste 5, atingiu-se o ponto mínimo em cerca de 200 iterações, um pouco menos eficiente que o teste anterior, porém, o ponto mínimo atingido foi também considerado muito próximo do ponto considerado ótimo, 0, este ponto atingido representa fx = 0.0162 de coordenadas (-0.0014; -0.0055).

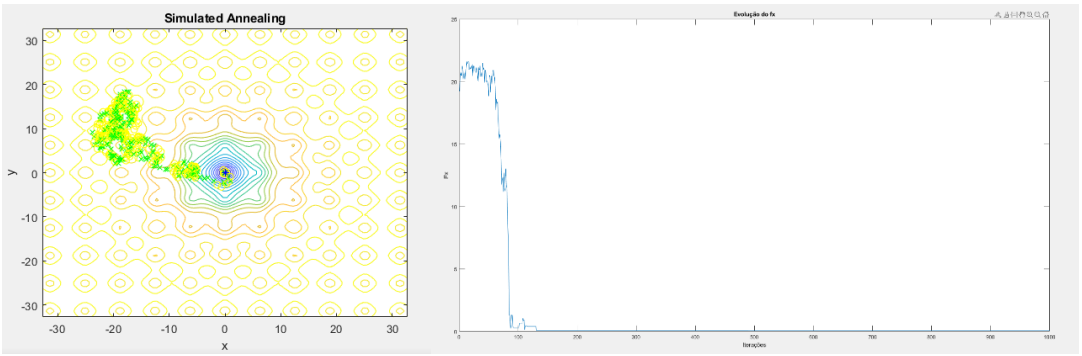

Figura 2.2.1.7 – Teste 6

Os resultados do teste final, 6, foram muito semelhantes aos do teste 5 (Figura 2.2.1.6), mas, o ciclo de pesquisa foi mais eficiente, tendo atingido o ponto ótimo em cerca de 140 iterações. O mínimo atingido foram as coordenadas (0.0153; 0.0126) em que o fx = 0.0562;

Para rematar estas análises, pode-se concluir que o Simulated Annealing é um bom algoritmo de pesquisa, mas deixa ainda a desejar, com resultados obtidos considerados não ideias, como são os exemplos dos testes 2 (Figura 2.2.1.3) e 3 (Figura 2.2.1.4). Mas são obtidos também resultados muito bons, todos eles entre 100-200 iterações, como os restantes resultados dos testes 1 (Figura 2.2.1.1), 4 (Figura 2.2.1.5), 5 (Figura 2.2.1.6) e 6 (Figura 2.2.1.7), onde um ponto muito próximo do considerado ótimo foi atingido.

2.2.2. Resolução do Problema com o Algoritmo Particle Swarm Optimization (PSO)

Tal como foi exposto previamente para o algoritmo do Simulated Annealing, foram realizados testes com o algoritmo do Particle Swarm Optimization para pesquisar o mínimo global da função no espaço de pesquisa, ou seja, a coordenada (0; 0) à qual equivale o valor fx = 0. Com a realização do primeiro teste, foram obtidos os seguintes resultados:

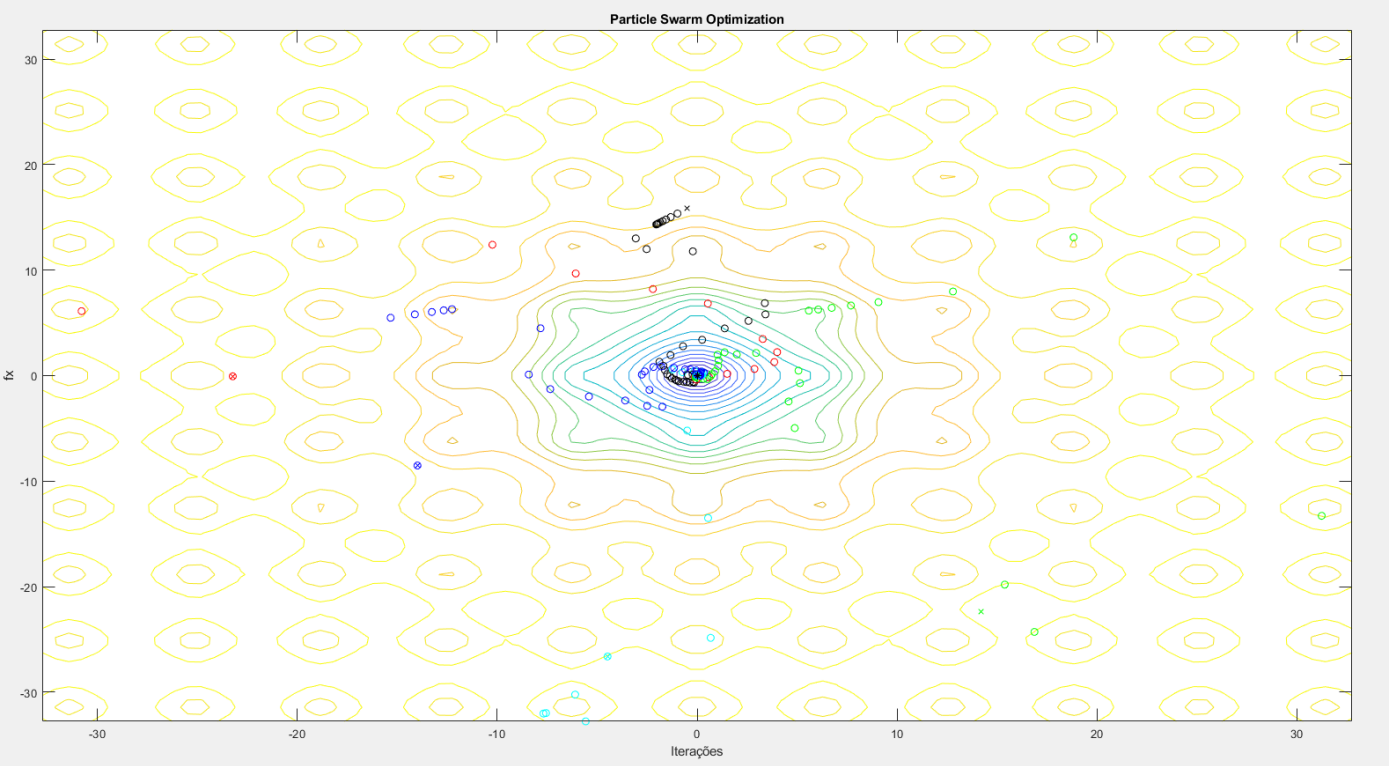

Figura 2.2.2.1. – Teste 1

Com este resultado, é possível observar que todas as partículas são inicializadas aleatoriamente (onde existe um símbolo ‘x’) e vão evoluindo conforme os seus conhecimentos cognitivos e sociais (representada pelo símbolo ‘o’). Tal como já foi verificado antes na explicação do algoritmo do PSO, as partículas vão formar um enxame no centro, porque “comunicam” umas com as outras e vão se aproximando todas umas das outras em direção ao centro. Analisando o teste 1 em maior pormenor:

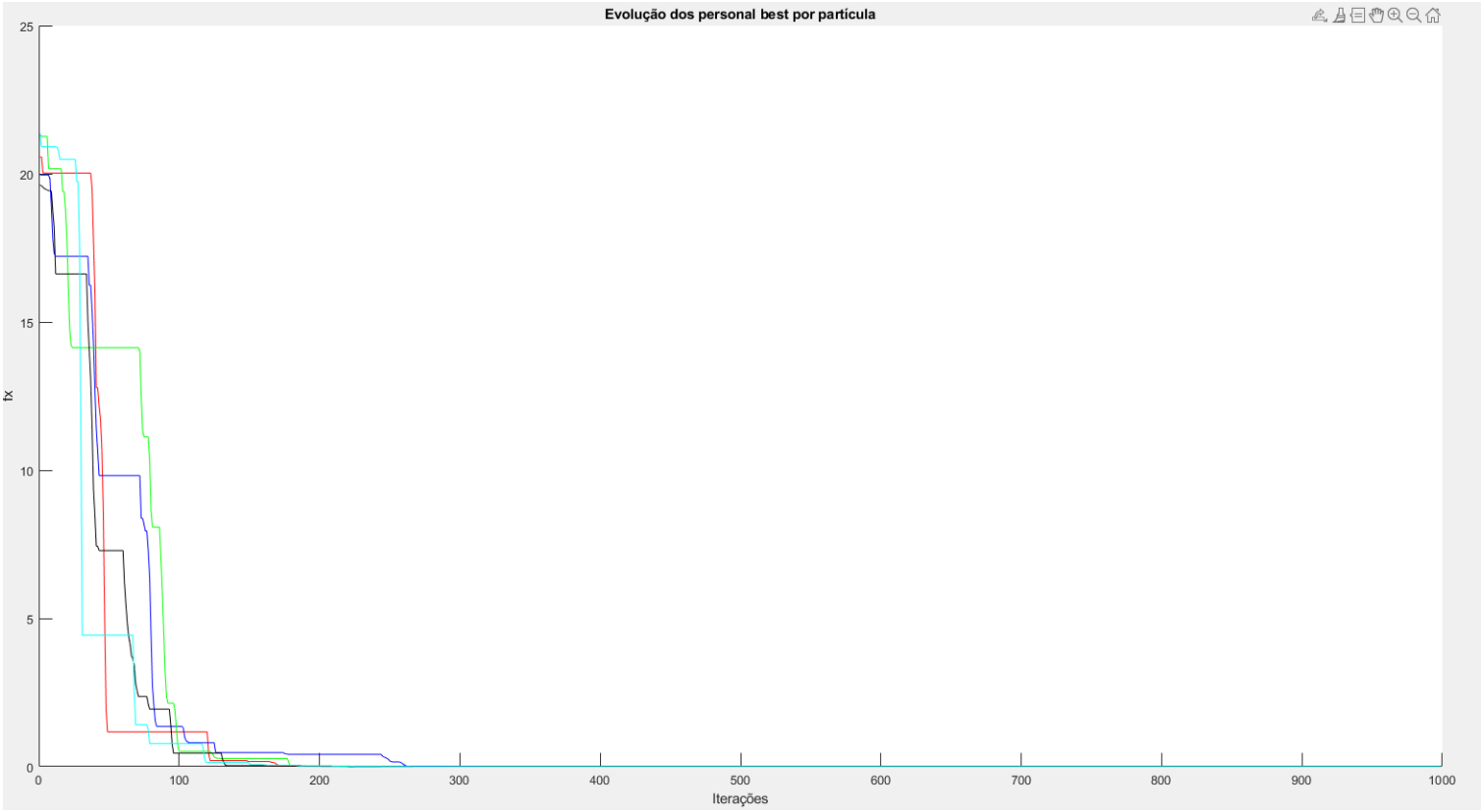

Figura 2.2.2.2 – Evolução dos personal best do teste 1

Na análise da evolução dos melhores pessoais de cada uma das partículas ao longo das iterações, pode ser analisado que todas vão tender para o mesmo valor, onde 4 partículas atingiram este valor à volta das 170 iterações e a última à volta das 260. O valor atingido foi considerado muito próximo do valor ótimo, 0, dada a coordenada (-0,00000000009; -0,00000000001) com valor fx = 0,00000000027, que é extremamento próximo do valor 0.

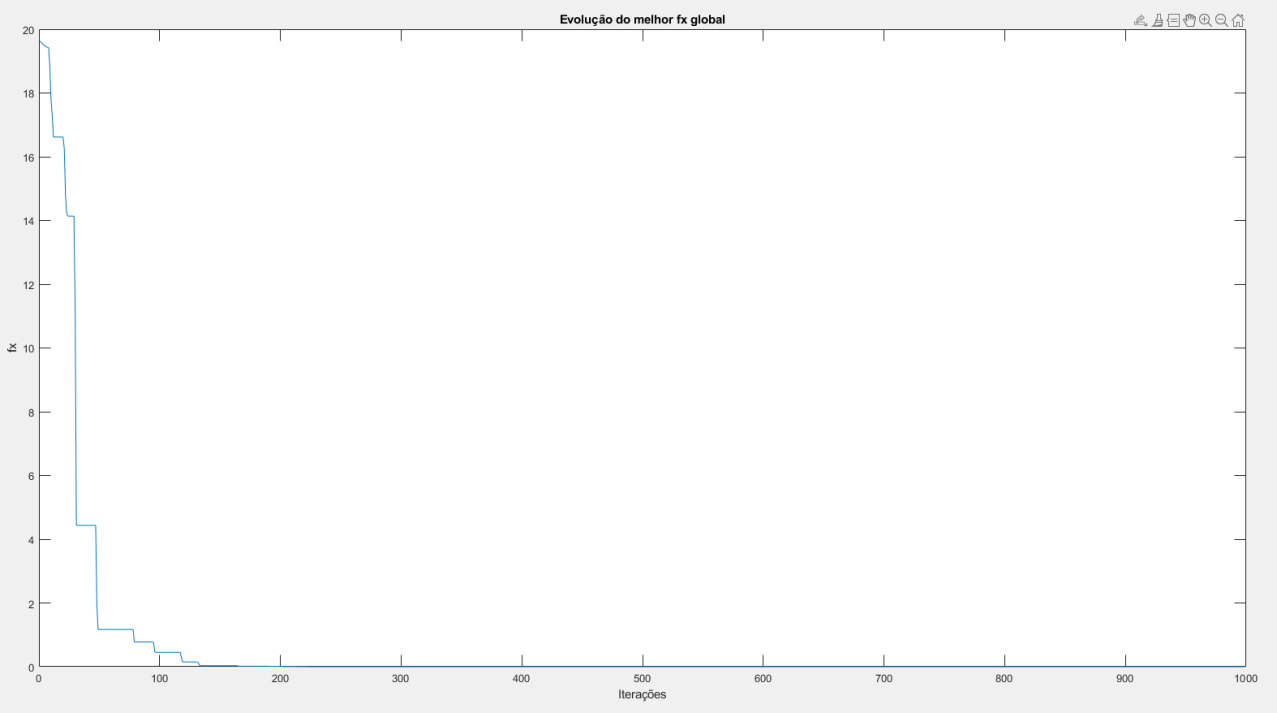

Figura 2.2.2.3 – Evolução do melhor fx global

Analisando o valor do melhor fx global, verifica-se que este foi atingido no final de cerca de 130 iterações, sendo este o valor já indicado previamente, para onde todas as partículas acabaram por convergir. No teste 2:

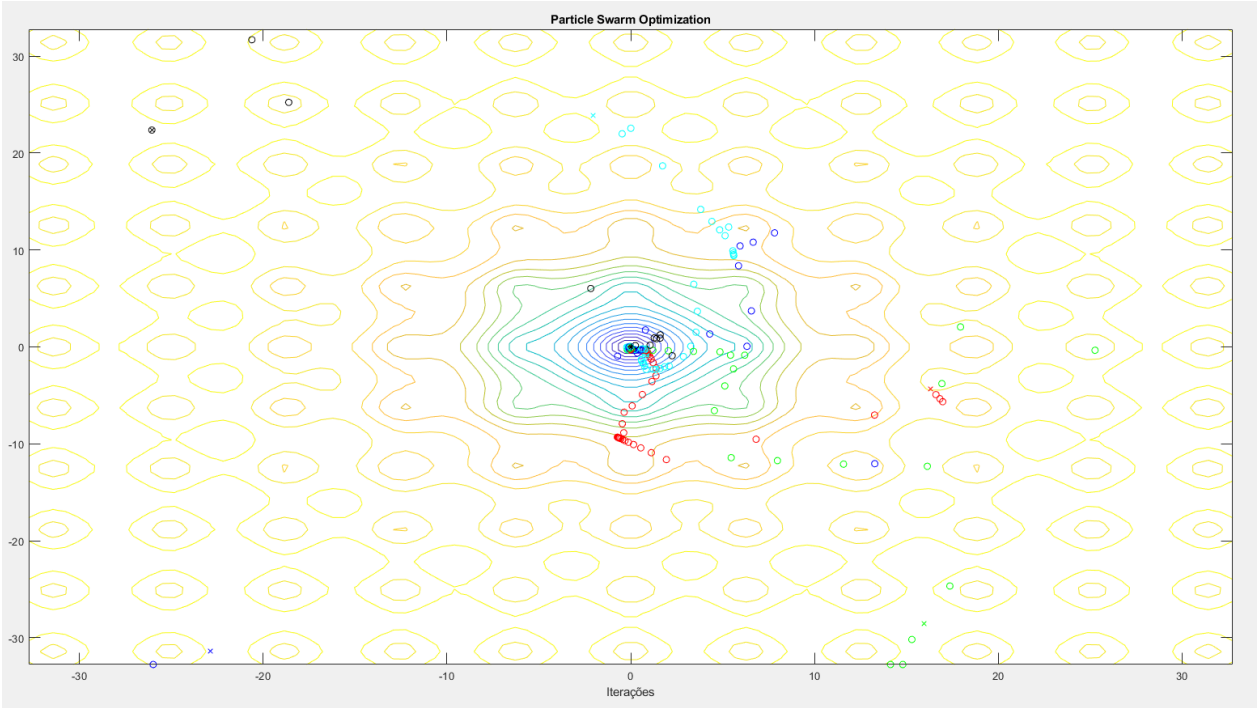

Figura 2.2.2.4 – Teste 2

Foi verificada também uma tendência para o mínimo global, apesar desta, neste teste, ser pior do que a do teste 1 (ver página 20). Neste teste, o mínimo atingido foi na coordenada (0,0000009; 0,0000005) onde o fx = 0,0017. Este resultado, apesar de ser pior que o anterior, foi muito bom pois foi perto do mínimo global, 0, que é o resultado perfeito.

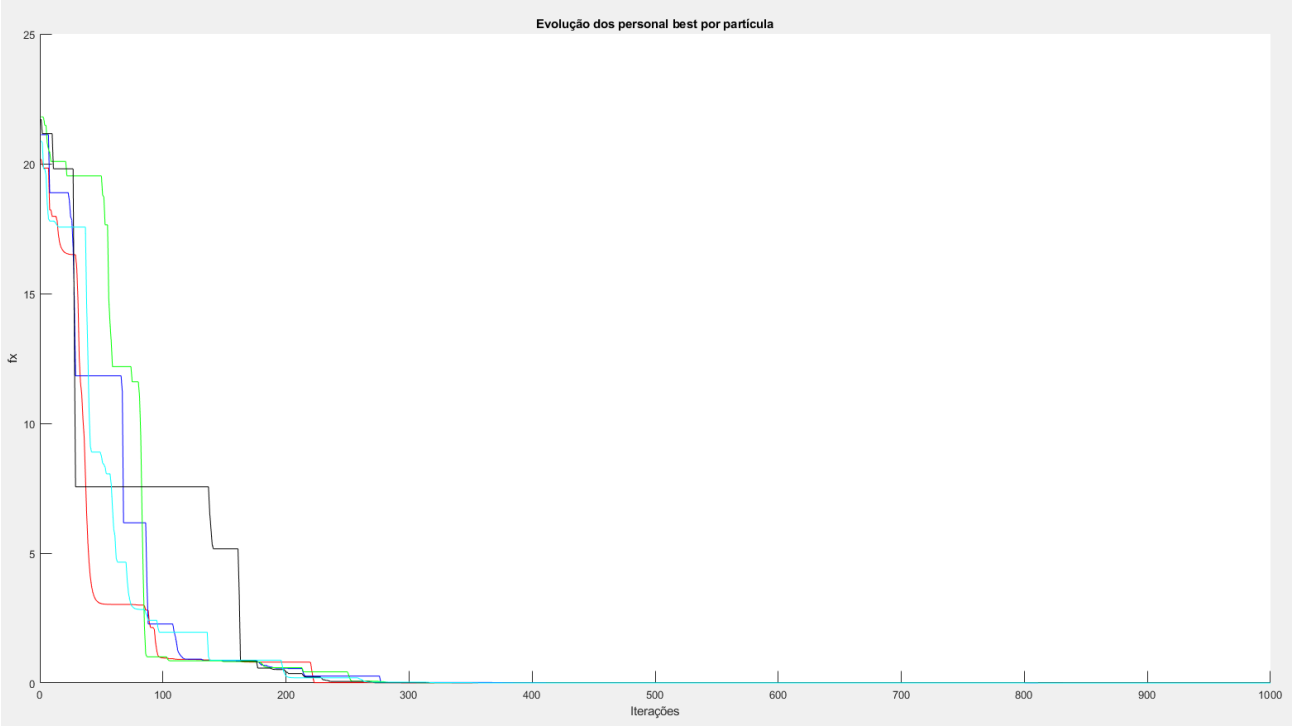

Figura 2.2.2.5 – Evolução dos personal best do teste 2

Neste teste, os personal best convergiram todos para o mínimo mencionado previamente no final de cerca de 270 iterações e a melhor posição global evoluiu de acordo com o seguinte gráfico:

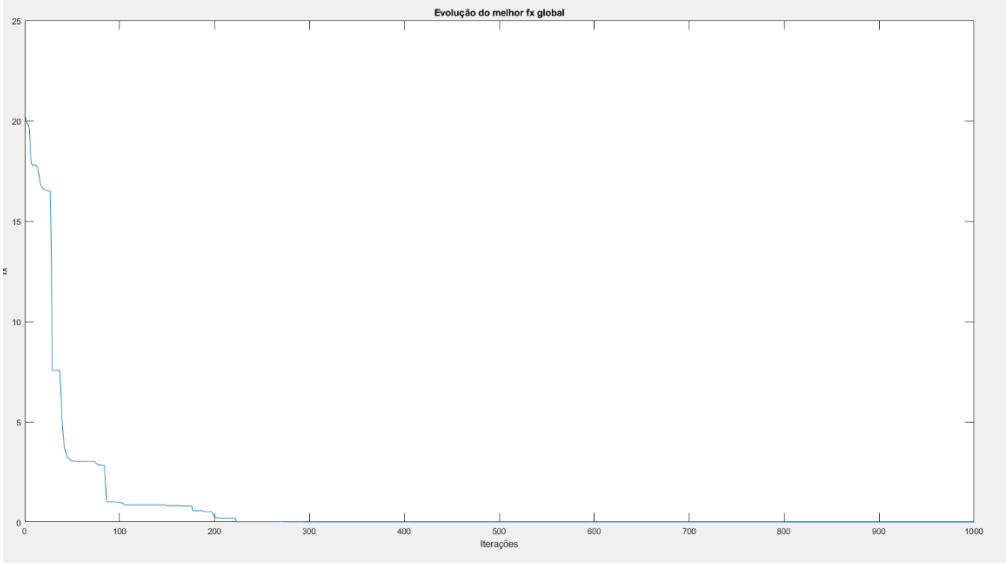

Figura 2.2.2.6 – Evolução do global best do teste 2

Podemos verificar que o melhor global foi alcançado ao final de cerca de 220 iterações.

No âmbito deste projeto foram realizados imensos testes, mas, para manter a brevidade do relatório, apenas serão mencionados alguns considerados mais relevantes, irá ser, portanto, apenas mostrado mais um ciclo de pesquisa, o teste 3:

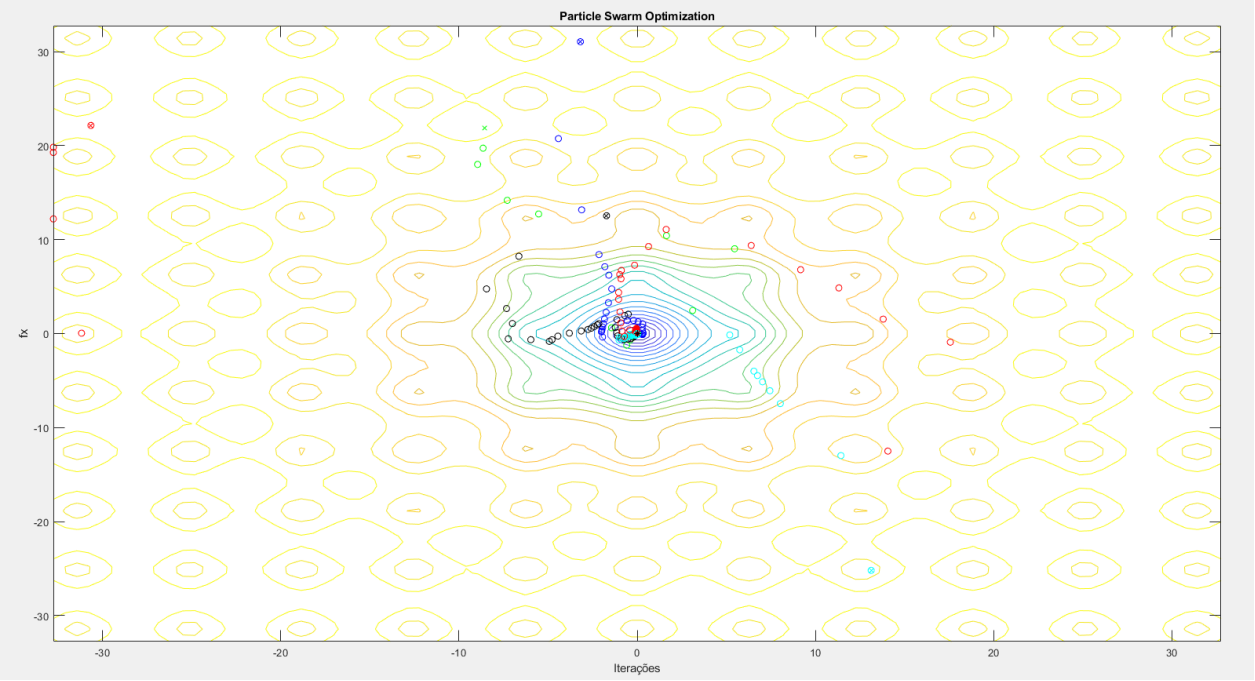

Figura 2.2.2.7 – Teste 3

Neste terceiro teste, foi alcançado também um ponto muito próximo do considerado ótimo. O ponto mínimo atingido foi de 0,00000004 com as coordenadas (-0,00000001; -0,00000004). Este resultado é muito próximo do mínimo global, sendo considerado um sucesso. Quanto à evolução dos fx pessoais:

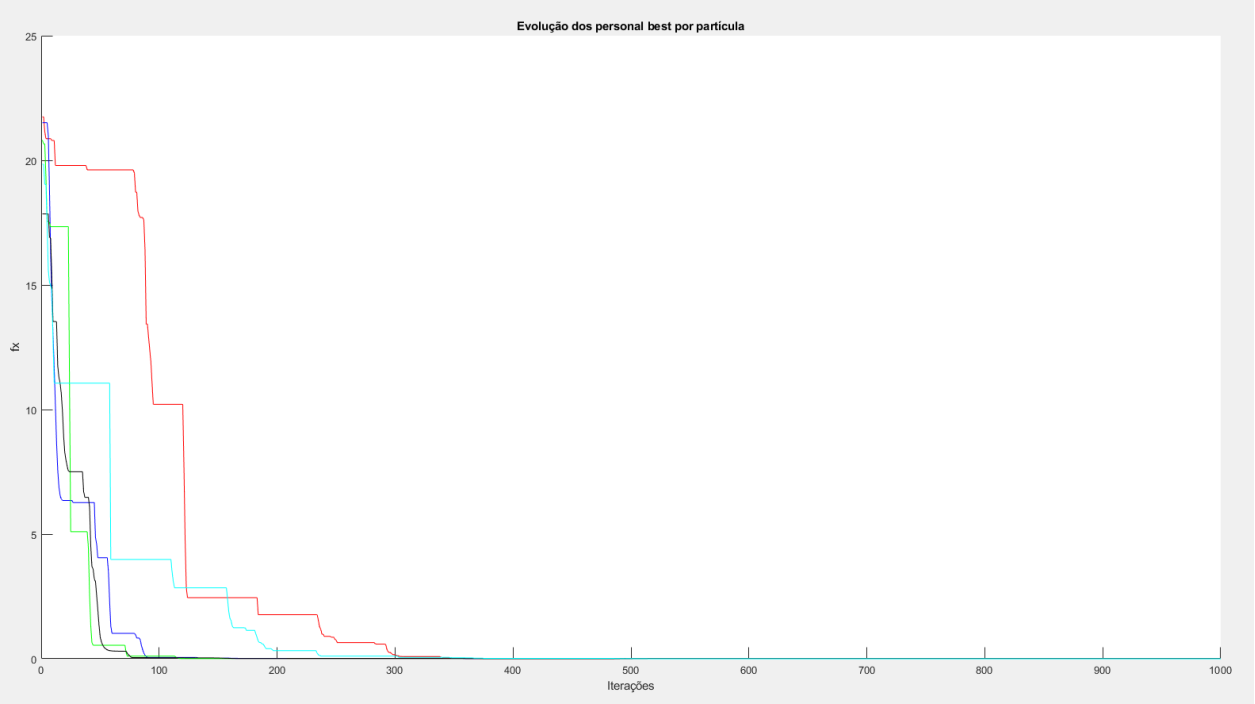

Figura 2.2.2.8 – Evolução dos personal best do teste 2

Pode ser verificado que todos os pontos convergiram para o mesmo resultado após cerca de 300 iterações, tendo os primeiros 3 convergido ao final de 100 iterações, o 4º ao final de 240 e o último apenas ao final de cerca de 300 iterações. Quanto à evolução do global best:

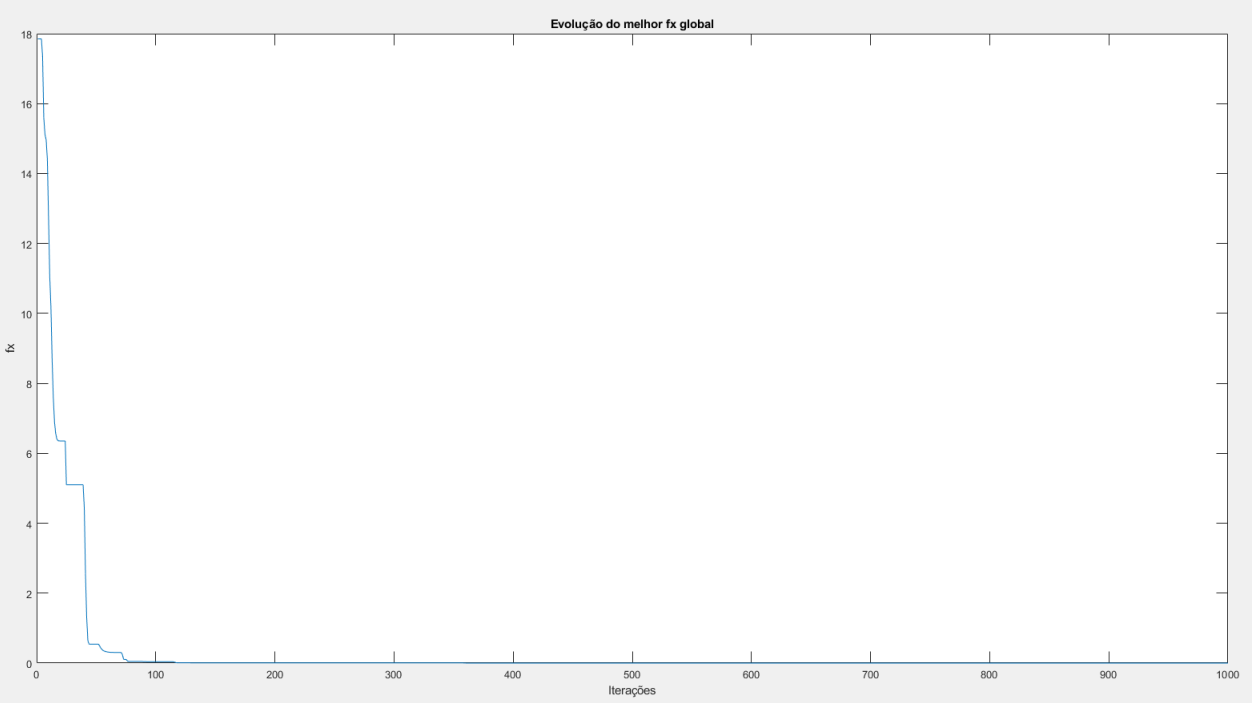

Figura 2.2.2.9 – Evolução do global best do teste 2

Verifica-se que, apesar do mínimo alcançado neste teste ter sido considerado pior que o mínimo alcançado, por exemplo, no teste 1, este demorou menos tempo a ser atingido, tendo demorado apenas cerca de 110 iterações a que este fosse atingido.

2.2.3. Comparação de Resultados dos Dois Algoritmos

Ao analisar todos os testes que foram realizados, incluindo os que foram expostos neste relatório para comparação, pode-se verificar um problema enorme do Simulated Annealing que o Particle Swarm Optimization não verifica. Porém, o Particle Swarm Optimization também tem uma desvantagem sob o Simulated Annealing, ao analisar-mos os resultados dos testes. Ao analisar-mos, por exemplo, o teste 4 do Simulated Annealing (ver página 18, Figura 2.2.1.5):

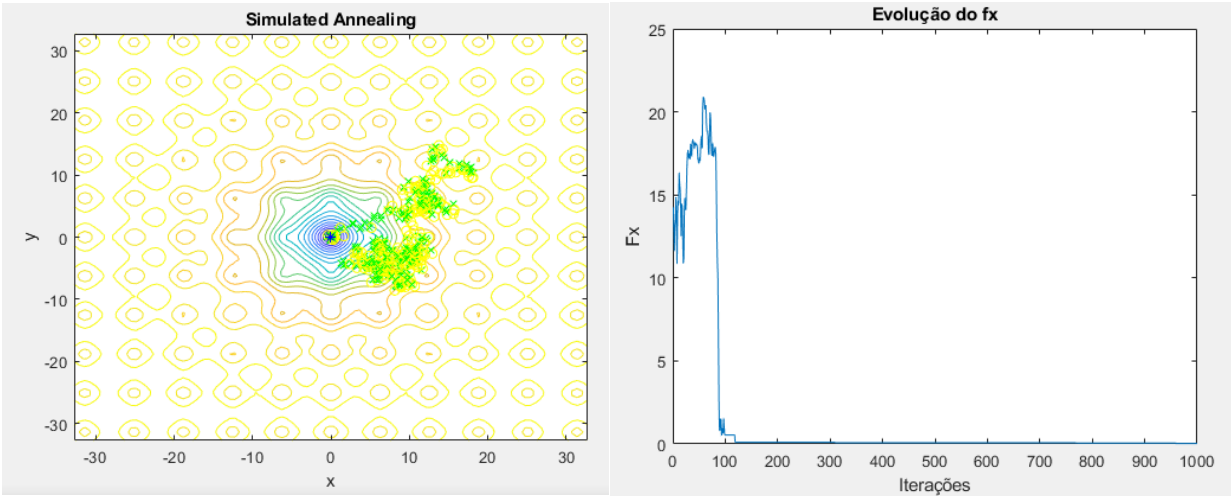

Figura 2.2.3.1 – Teste 4 do SA

verificamos que este atingiu o ponto mínimo apenas ao final de cerca de 110 iterações, enquanto que, ao analisar a grande maioria dos resultados obtidos por parte do Particle Swarm Optimization, como, por exemplo, o teste 2 (ver página 23, Figura 2.2.2.6):

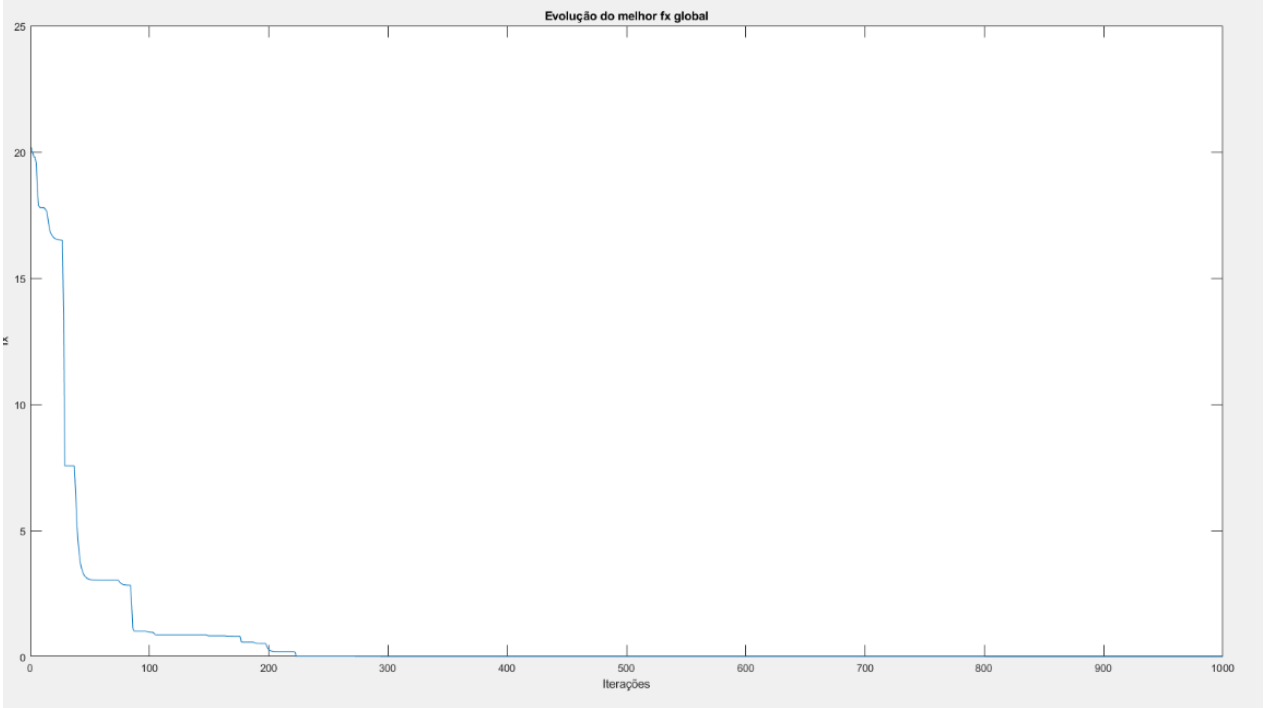

Figura 2.2.3.2 – Teste 2 do PSO

pode-se verificar que, na maioria dos casos, apesar do mínimo ótimo ser quase atingido, o PSO tende a ter um custo maior que o Simulated Annealing, isto é, requer, por norma, mais iterações até que o resultado pretendido seja alcançado.

Porém, o Simulated Annealing possui uma desvantagem massiva em comparação com o algoritmo do PSO. Ao analisar, por exemplo, o teste 3 do SA (ver página 18, Figura 2.2.1.4):

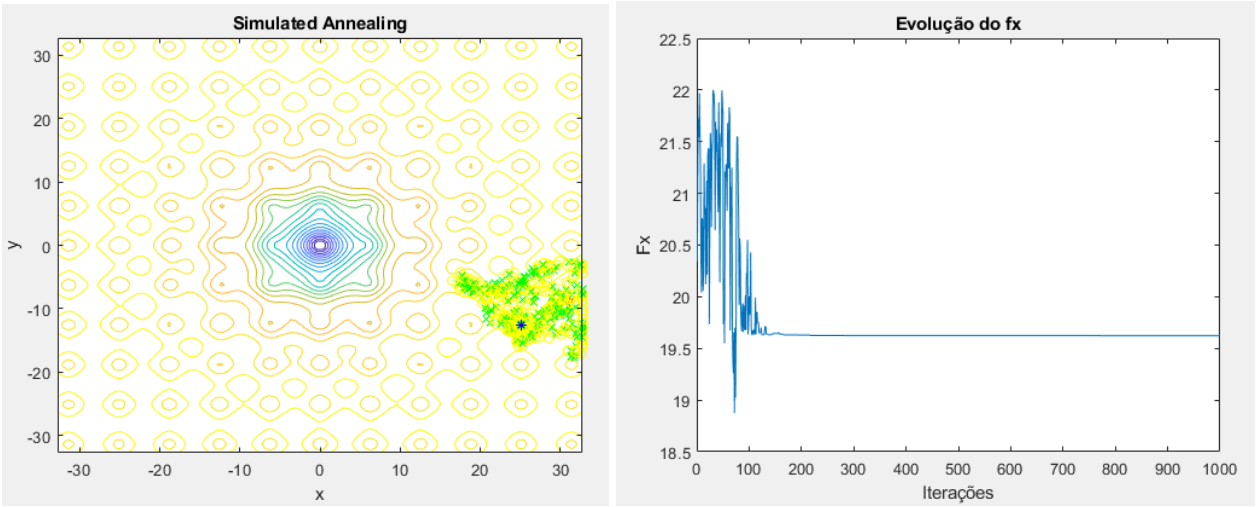

Figura 2.2.3.3 – Teste 3 do Simulated Annealing

verifica-se uma lacuna enorme neste algoritmo. Este tem a possibilidade, negativa, de ficar "preso" em mínimos locais, o que não é o pretendido. Esta é uma vantagem que o PSO tem sobre o Simulated Annealing. Em todos os testes que foram realizados para a realização deste projeto, em nenhum o do Particle Swarm Optimization ficou "preso" num mínimo local.

Resumindo:

PSO:

Vantagens:

- Alcança sempre o objetivo pretendido;
- Algoritmo forte para resolução de problemas;

Desvantagens:

- Custo de pesquisa mais elevado que alguns outros algoritmos;

SA:

Vantagens:

- Baixo custo de pesquisa;

Desvantagens:

- Pode ficar "preso" num mínimo local, o que não é pretendido;

O PSO é considerado, por estas razões, mais viável do que o Simulated Annealing, porém, com um custo mais elevado.

3. Conclusão

Os métodos de pesquisa são muito valiosos no dia-a-dia do planeta. Servem para encontrar aquilo que um agente procura. O Particle Swarm Optimization (PSO) utiliza informações de bando, para que as partículas “comuniquem” umas com as outras e encontrem, em grupo, uma solução que agrade a todos. O Simulated Annealing baseia-se na técnica dos ferreiros denominada Annealing, que aceita soluções piores que a encontrada previamente, mas, apenas até um certo ponto, até que a temperatura atinja 0, onde apenas soluções melhores serão aceites.

Ao abordar mais intensivamente os algoritmos desenvolvidos do Particle Swarm Optimization (PSO) e Simulated Annealing, ficou mais esclarecido os propósitos destes dois métodos de pesquisa meta-heurísticos.

Após análise minuciosa de todas as etapas, backgrounds, e testes de cada um dos algoritmos de pesquisa abordados neste projeto, o algoritmo de pesquisa do PSO possui uma vantagem enorme sob o do Simulated Annealing, sendo, para fins deste relatório, considerada superior ao do Simulated Annealing, pois é fortemente mais viável do que o do Simulated Annealing, alcançando, em média, um ponto, no espaço de pesquisa, mais próximo do pretendido do que o do Simulated Annealing. O algoritmo do Simulated Annealing não deixa de ser útil, por exemplo, para problemas menos complexos do que o do PSO, pois, apesar de ser menos viável que o do PSO, possui um custo mais reduzido que este, bom para quando o tempo de procura de uma solução é reduzido e a complexidade do problema não requer tantas restrições.

4. Bibliografia

[1] Perdicoúlis, A. (2014) References. Technical Folios, 4

[2] Paulo Moura Oliveira, Eduardo Solteiro Pires (2020) Protocolo de Trabalho Prático: Métodos de Pesquisa

[3] Paulo Moura Oliveira, Eduardo Solteiro Pires (2020) Modelo para o Relatório dos Trabalhos

[4] Paulo Moura Oliveira, Nature Inspired Search and Optimization: a simplified approach, Part I

[5] Paulo Moura Oliveira, (2020), Métodos de Pesquisa (Search) – Part II

[6] Paulo Moura Oliveira, (2020), Métodos de Pesquisa (Search) – Part IV

[7] Paulo Moura Oliveira, (2018), Simulated Annealing (Em Português), https://www.youtube.com/watch?v=w2rBcPo88XM&feature=youtu.be
Acedido em 27-12-2020

[8] Paulo Moura Oliveira, (2018), Particle Swarm Optimization Algorithm (PSO) (Em Português), https://www.youtube.com/watch?v=Ybn-_6KsxCA&feature=youtu.be
Acedido em 30-12-2020

[9] Paulo Moura Oliveira, (2020), Particle Swarm Optimization (PSO) (In English), https://www.youtube.com/watch?v=EOHWWdy2iQ4&feature=youtu.be
Acedido em 2-1-2021

Anexo A – Código Simulated Annealing

```
quality = @(x1,x2) -20*exp(-0.2*sqrt(1/2*(x1.^2+x2.^2)))-
exp(1/2*(cos(x1)+cos(x2)))+ 20 + exp(1); %Função fornecida

%Estabelecimento dos limites e design do gráfico%%%
x1 = linspace(-32.768,32.768,100);                %
x2 = linspace(-32.768,32.768,100);                %
[X1,X2] =  meshgrid(x1,x2);                       %
Fx = quality(X1,X2);                              %
contour(X1,X2,Fx,20);                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

T = 90; %Inicialização da variável temperatura (para proceder-mos a uma
analogia entre a temperatura de um metal
        %e este parâmetro ajustável do algoritmo, visto que este se baseia
numa analogia deste tipo

r = (rand(1,2)-0.5)*2*32.768; %Gera uma matriz de 1 por 2 de números
aleatórios entre -32.768 e 32.768

fx = quality(r(1), r(2)); %Cálculo do valor da qualidade da função segundo os
valores gerados aleatóriamente

hold on %Retém o gráfico atual para que novos plots sejam realizados no mesmo
gráfico do inicial

plot(r(1), r(2), 'r*') %Coloca a partícula no gráfico
xlabel('x') %Label do eixo dos x
ylabel('y') %Label do eixo dos y
title('Simulated Annealing') %Título do gráfico

n = 0; %t e n são variáveis inicializadas para que possamos estabelecer o
número de iterações do ciclo
t = 1;

evo(t) = 0; %Inicialização da variável evo para verificarmos a evolução da
qualidade da função, isto é, do fx
temp(t) = 0; %Inicialização da variável temp para verificarmos a evolução da
temepratura, isto é, de T
probabilidade(t) = 0; %Inicialização da variável para verificarmos a evolução
da probabilidade, isto é, do prob

while(t <= 1000) %Inicialização do ciclo while para que o algoritmo funcione

    n = 0; %Reinicialização da variável n para que o ciclo while seguinte seja
percorrido 5 vezes por cada
           %vez que o ciclo while em que este se insere percorre

    while(n <= 5) %Inicilização de um segundo ciclo while

        r_new = r + (rand(1,2)-0.5)*2*1; %Geração de novas variáveis
aleatórias na vizinhança das precedentes (r)

        r_new = max(min(r_new,32.768),-32.768); %Estabelecimento de um limite
na geração das novas variáveis
```

```
                                                    %para que estas não
ultrapassem os limites do gráfico e não saiam
                                                    %do domínio deste

        fx_new = quality(r_new(1), r_new(2)); %Cálculo da qualidade da nova
partícula que irá ser potencialmente gerada

        deltaE = fx_new - fx; %Cálculo da variância da energia ao longo do
ciclo

        prob = exp(-deltaE/T); %Cálculo da probabilidade de aceitação da nova
partícula, que irá ser menor conforme
                               %o número de iterações

        prob = max(min(prob,1),0); %Para que a probabilidade não ultrapasse 1,
pois isso é matematicamente impossível

        if deltaE < 0 %Caso a amplitude das energias seja menor que 0, a nova
partícula é automaticamente aceite e a
                      %anterior substituída pela nova

            r = r_new; %As coordenadas aleatórias geradas anteriormente são
substituídas pelas da nova partícula
            fx = fx_new; %A qualidade, ou fx, da partícula anterior é
substituída pela da nova partícula
            plot(r(1), r(2), 'gx') %A nova partícula é colocada no gráfico

        elseif rand < prob %Caso a condição anterior não seja verificada, é
feita uma nova verificação e garantida uma
                           %segunda oportunidade para que a partícula nova
possa ser aceite. Se a probabilidade,
                           %calculada anteriormente, for maior que um número
gerado  aleatóriamente entre 0 e 1,
                           %esta nova partícula, com fx maior que aanterior, o
que não é o inicialmente pretendido,
                           %é aceite e substitui a anterior

            r = r_new; %As coordenadas aleatórias geradas anteriormente são
substituídas pelas da nova partícula
            fx = fx_new; %A qualidade, ou fx, da partícula anterior é
substituída pela da nova partícula
            plot(r(1), r(2), 'yo') %A nova partícula é colocada no gráfico

        end %Fim da constraint if

            evo(t) = fx; %No final de cada ciclo, na matrix evo é registada a
qualidade da partícula
            n = n + 1; %A cada ciclo n incrementa 1 valor para que o ciclo
while funcione corretamente

    end %Fim do ciclo while interno

       if t < 101 %Constraint para que só seja armazenada até à iteração 100
uma vez que a temperatura não decai mais
       temp(t) = T; %Registo da temperatura a cada ciclo, para que possa ser
observada a sua evolução
       end %Fim do if
```

```
        probabilidade(t) = prob; % Registo da evolução da probabilidade a cada
ciclo, para ser analisada

        T = 0.94*T; %Decaimento da temperatura ao longo do ciclo while, com um
fator de 0,94

        t = t + 1; %A cada ciclo t incrementa 1 valor para que o ciclo while
funcione corretamente

end %Fim do ciclo while externo

plot(r(1),r(2),'b*') %Partícula melhor, ou seja, com o menor valor de fx
ecentrada é colocada no gráfico
                     %com uma cor diferente das outras, para que possa ser
facilmente identificada como a melhor

hold off %Liberta o gráfico que estava atualmente retido

figure %Figure cria uma nova janela com uma nova figura, no nosso caso um
gráfico para que possamos inserir os dados
plot(evo) %Insere os dados da evolução do fx
ylabel('Fx') %Label do eixo dos y
xlabel('Iterações') %Label do eixo dos x-
title('Evolução do fx') %Título do gráfico

figure
plot(temp) %Insere os dados da evolução da temperatura
ylabel('T') %Label do eixo dos y
xlabel('Iterações') %Label do eixo dos x
title('Evolução da temperatura') %Título do gráfico

figure
plot(probabilidade) %Insere os dados da evolução da probabilidade
ylabel('Probabilidade') %Label do eixo dos y
xlabel('Iterações') %Label do eixo dos x
title('Evolução da probabilidade') %Título do gráfico
```

Anexo B – Código Particle Swarm Optimization

```
quality = @(x1,x2) -20.*exp(-0.2*sqrt(1/2*(x1.^2+x2.^2)))-
exp(1/2*(cos(x1)+cos(x2)))+ 20 + exp(1); %Função fornecida

%Estabelecimento dos limites e design do gráfico%%%
x1 = linspace(-32.768,32.768,100);                %
x2 = linspace(-32.768,32.768,100);                %
[X1,X2] =  meshgrid(x1,x2);                       %
Fx = quality(X1,X2);                              %
contour(X1,X2,Fx,20);                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

hold on %Retém o gráfico atual para que novos plots sejam realizados no mesmo
gráfico do inicial

x = (rand(5,2)-0.5)*2*32.768; %Matriz de coordenadas aleatórias

fx = quality(x(:,1), x(:,2)); %Quality inicial de cada partícula

plot(x(1,1), x(1,2), 'rx') %Colocar as 5 particulas inicializadas
aleatoriamente num gráfico
plot(x(2,1), x(2,2), 'bx')
plot(x(3,1), x(3,2), 'gx')
plot(x(4,1), x(4,2), 'kx')
plot(x(5,1), x(5,2), 'cx')

xlabel('Iterações'); %Eixo dos x e dos y assim como o título do gráfico são
definidos aqui
ylabel('fx');
title('Particle Swarm Optimization');

b = x; %Inicialização da matriz de coordenadas com os personal best de cada
partícula
fb = fx; %Inicialização da matriz das qualidades personal best de cada
partícula

[fg, i] = min(fb); %Inicialização da melhor quality global, isto é, o min
encontra o valor mínimo da matriz
                   %com os melhores fx pessoais de cada partícula, o melhor é
atribuído ao valor fg
g = b(i,:); %Inicialização das coordenadas global best

v = (rand(5,2)-0.5)*2*1; %Velocidade inicial de cada particula gerada
aleatoriamente entre 0 e 1

n = 1; %Inicializador do loop while
w = 0.7; %Fator de inércia

bestred(n) = 0;      %Inicialização das variáveis para armazenar os personal e
globalbests de
bestblue(n) = 0;     %cada partícula para futura referência
bestgreen(n) = 0;
bestblack(n) = 0;
bestcyan(n) = 0;
globalbest(n) = 0;
```

```
while n <= 1000 %Inicialização do ciclo while, calculado para que w seja 0.4
no final do ciclo

    phic = (rand(5,2)-0.5)*2*1; %Inicialização do phi cognitivo, erro
cognitivo, utilizado na equação de
                                %evolução de velocidade da partícula
    phis = (rand(5,2)-0.5)*2*1; %Inicialização do phi sociial, erro social,
utilizado na equação de
                                %evolução de velocidade da partícula

    w = w - 0.0003; %Evolução do fator de inércia ao longo do ciclo

    v = w * v + 0.2 * phic.*(b - x) + 0.2 * phis.*(g - x); %Equação de
determinação de velocidade das partículas
                                                           %ao longo do ciclo,onde
a velocidade aumenta com cada
    %Nota: foram considerados os valores 1.5 para      %loop onde w representa
o fator de inércia, v a
    %      os valores de c1 e c2 porque, devido ao     %velocidade anterior,
1.5 é o c1 ou seja, a
    %      baixo número de iterações, com o valor      %componente de peso
para o fator de erro cognitivo,
    %      2 iriam raramente todas as partículas       %phic é o erro
cognitivo, (b-x) significa o valor
    %      converger para o mesmo valor                %das coordenadas
melhores de cada partícula menos o
                                                       %valor gerado
aleatoriamente, o segundo 1.5
                                                       %representa o c2, ou
seja um peso para o fator social,
                                                       %phis e (g-x) são as
coordenadas melhores
                                                       %globais menos as
coordenadas geradas aleatoriamente

    x = x + v; %Equação de atualização das coordenadas de cada partícula, onde
x representa a matriz das
               %coordenadas de cada uma das partículas e v a velocidade
determinada na equação anterior

%Estas duas equações representam o coração do PSO, isto é, a partir destas
duas equações é que o PSO funciona corretamente

    x = max(min(x,32.768),-32.768); %Estabelecimento de um limite no cálculo
das novas coordenadas para que estas não
                                    %ultrapassem os limites do gráfico e não
saiam do domínio deste

    fx = quality(x(:,1), x(:,2)); %Determinação do novo fx de cada partícula,
ou seja, da qualidade nova de cada partícula

    for i = 1:1:5 %Ciclo for para determinar se o valor atual verifica um
valor menor do que o previamente estipulado
    if fx(i) < fb(i) %Caso isto seja verdade:
        fb(i) = fx(i); %O valor anterior no local (i, 1) da matriz será
substituído pela nova coordenada
```

```
        b(i,1) = x(i,1); %A coordenada anterior no local (i, i) da matriz será
substituída pela nova melhor coordenada
        b(i,2) = x(i,2);
    end %Fim do ciclo if
    end %Fim do ciclo for

    [fg, i] = min(fb); %Determinação do melhor valor (quality, fx) global
    g = b(i,:); %Determinação da melhor coordenada global

    plot(b(1,1), b(1,2), 'ro') %Inserção no gráfico de cada uma das partículas
conforme encontram melhores pessoais
    plot(b(2,1), b(2,2), 'bo')
    plot(b(3,1), b(3,2), 'go')
    plot(b(4,1), b(4,2), 'ko')
    plot(b(5,1), b(5,2), 'co')

    bestred(n) = fb(1);  %Armazenamento das melhores globais e pessoais a cada
iteração para futura referência
    bestblue(n) = fb(2);
    bestgreen(n) = fb(3);
    bestblack(n) = fb(4);
    bestcyan(n) = fb(5);
    globalbest(n) = fg;

    n = n + 1; %A cada ciclo n incrementa 1 valor para que o ciclo while
funcione corretamente

end %Fim do ciclo while

plot(g(1),g(2), 'k*'); %Plot da melhor variável global encontrada, ou seja, o
valor mínimo encontrado

hold off %Liberta o gráfico que estava atualmente retido

figure %Gráfico à parte para visualizar-mos a evolução do melhor fx global ao
longo das iterações, o fg
plot(globalbest);
xlabel('Iterações');
ylabel('fx');
title('Evolução do melhor fx global');

figure %Gráfico à parte para visualizarmos a evolução dos melhores fx pessoais
ao longo das iterações, os fb
hold on
plot(bestred, 'r')
plot(bestblue, 'b')
plot(bestgreen, 'g')
plot(bestblack, 'k')
plot(bestcyan, 'c')
xlabel('Iterações');
ylabel('fx');
title('Evolução dos personal best por partícula');
hold off
```