Otimização de Funções Através de Métodos de Pesquisa

Engenharia Informática

Inteligência Artificial

2º Trabalho Prático

07/12/2022

Por:
António Teixeira, 70835

Índice

Abstract

Este relatório examina as funções de diversos algoritmos de pesquisa fundamentados pela inteligência artificial.

Esta é uma área que tem transformado as operações industriais.

Uma das mais importantes aplicações é reduzir o custo computacional de otimização.

Keywords: Hill Climbing, Simulated Annealing, Multiple Restart Hill Climbing, Multiple Restart Simulated Annealing, Random Walk

Introdução

“Pretende-se promover a aquisição de conhecimentos e desenvolvimento de competências relativas aos seguintes algoritmos: Subida da colina estocástica e Simulated Annealing” - protocolo do trabalho prático 2.

No presente relatório será descrito o funcionamento, tal como a criação dos algoritmos Hill Climbing, Simulated Annealing e Random Walk.

MATLAB, abreviação de “MATrix LABoratory”, é uma linguagem de programação multiparadigma proprietária e ambiente de computação numérica desenvolvido pela MathWorks. MATLAB permite a manipulação de matrizes, desenho de funções e dados, criação de algoritmos e interfaces:

Neste trabalho foi solicitada a criação e comparação de protótipos de algoritmos iterativos, tais como:

Hill Climbing;
Multiple Restart Hill Climbing;
Simulated Annealing;
Multiple Restart Simulated Annealing;
Random Walk;

Enquadramento Teórico

➢ Hill Climbing

Em análise numérica, *hill climbing*, é uma técnica de otimização matemática pertencente à família da "procura local". Este é um algoritmo iterativo que começa com uma solução arbitrária para um problema e de seguida tenta encontrar uma melhor solução através de mudanças incrementais à solução. Se esta mudança produzir uma melhor solução, é feita outra mudança incrementar, até que nenhuma outra melhoria seja encontrada.

Hill Climbing, encontra soluções ótimas para problemas convexos, para outro tipo de problemas encontrará apenas soluções localmente, tais não são necessariamente as melhores soluções - "ótimo global".

De modo a evitar-se o confinamento num ótimo local, é possível a reinicialização - tal será abordado posteriormente.

A simplicidade do algoritmo torna-o uma popular primeira escolha entre algoritmos de otimização. É então usado vastamente em inteligência artificial, de modo a alcançar um objetivo através de um ponto de partida.

Relativamente à descrição matemática, *hill climbing*, tenta maximizar, ou minimizar, uma função alvo $\mathbf{f(x)}$, onde $\mathbf{x}$ é um vetor de valores contínuos ou discretos. A cada iteração o algoritmo ajustará um elemento singular em $\mathbf{x}$ e determinará se a mudança melhora $\mathbf{f(x)}$. Qualquer mudança que melhora $\mathbf{f(x)}$ é aceite, e o processo repete-se até não ser possível, através de qualquer mudança, melhorar o valor de $\mathbf{f(x)}$. Diz-se $\mathbf{x}$ "otimizado localmente".

➢ Simulated Annealing

Simulated Annealing é uma técnica probabilística de aproximação de um “ótimo global” de uma dada função. Especificamente, é uma meta heurística [1] de aproximação num grande espaço.

O nome do algoritmo provém da metalúrgica, “*annealing*”, uma técnica que envolve aquecimento e arrefecimento controlado de um material de modo a alterar as suas propriedades físicas.

Simulated Annealing é normalmente utilizado quando o espaço de procura é discreto e apresenta vantagem em problemas onde encontrar uma aproximação de um “ótimo global” é mais importante que encontrar um “ótimo local” preciso.

A noção de arrefecimento controlado criado no *simulated annealing* é interpretado como um lento decréscimo relativo à probabilidade de aceitar piores soluções à medida de que o espaço é explorado. Tal permite uma procura mais extensa.

A temperatura decresce progressivamente, partindo de um valor inicial positivo para zero. A cada passo, o algoritmo seleciona aleatoriamente uma solução próxima da solução atual, mede a qualidade, e move-se em direção a esse de acordo com a probabilidade, dependente da temperatura, de aceitar inferiores melhores ou piores soluções, a qual começa em 1 e diminui até 0.

Sendo por vezes necessário retornar a uma solução consideravelmente melhor, ao invés de efetuar aproximação utilizando sempre o estado atual, é também possível a reinicialização.

Resultados

Foram analisados os diferentes algoritmos para:

$$a\left(1-x^2\right)e^{-x^2-(1+y)^2}-b\left(\frac{x}{5}-x^3-x^5\right)e^{-x^2-y^2}+\left(-\frac{1}{3}\right)e^{-(x+1)^2-y^2}$$

, a = 3 e b = 10

1. Definiu-se a função e foi desenhado o gráfico com um ponto aleatório pertencente aos limites da mesma.

```
% Function Definition

a = 3;
b = 10;

% f1xy = @(x, y)    a.*(1-x.^2).*exp(-x.^2 - (1 + y).^2);
% f2xy = @(x, y)    -b.*(x./5 - x.^3 - x.^5).*exp(-x.^2 - y.^2);
% f3xy = @(x, y)    (-1/3)*exp(-(x + 1).^2 - y.^2);

% ftxy = f1xy(x, y) + f2xy(x, y) + f3xy(x, y);

% xy_max = [3 3];
% xy_min = [-3 -3];

% Definition and Limits

ftxy = @(x, y)  a.*(1-x.^2).*exp(-x.^2 - (1 + y).^2) -b.*(x./5 - x.^3 - x.^5).*exp(-x.^2 - y.^2) + (-1/3)*exp(-(x + 1).^2 - y.^2);

x = linspace(-3, 3, 100);
y = linspace(-3, 3, 100);

% Rectangular Grid

[X, Y] = meshgrid(x, y);

ftXY = ftxy(X, Y);

contour(X, Y, ftXY, 20)

% Random Coordenates

rx = (rand - 0.5)*2*3;
ry = (rand - 0.5)*2*3;

% Graph

hold on
plot(rx, ry, 'ro')

hold off
```

2. Foram definidos os limites das coordenadas **x** e **y**.

```
%

x_min = -3;
x_max = 3;
y_min = -3;
y_max = 3;
```

3. Foi definida a quantia máxima, e mínima a avançar a cada iteração.

```
% Increment Limits

min_sum = -0.03;
max_sum = 0.03;
```

➢ Hill Climbing

1. Em primeiro lugar, foram definidas o número de iterações e vetores inicializados correspondentes às coordenadas **x** e **y**.

```
% History of Coordenates

maxiterations = 1000;

x_coordenates = zeros(maxiterations, 1);
y_coordenates = zeros(maxiterations, 1);
```

2. Foi calculado um ponto inicial aleatoriamente. Tal foi marcado no gráfico da função como um asterisco vermelho.

```
% Random Coordenates

x = (x_max - x_min)*rand + x_min;
y = (y_max - y_min)*rand + y_min;

hold on
plot(x, y, 'r*')
```

3. Foram estabelecidas as primeiras posições dos vetores inicializados como **x** e **y** desse ponto calculado. Tal como definida a primeira posição do vetor de resultados e "ótimos" **x** e **y**.

```
% Initial Values and Variables

x_coordenates(1) = x;
y_coordenates(1) = y;

txy(1) = ftxy(x, y);

best_x = x;
best_y = y;
```

4. Foi inicializado o “ciclo *for*”, a ocorrer de início em 2 e até ao máximo número de iterações. Foram também calculados novos **x** e **y** com base nos obtidos anteriormente.

```
for k = 2:maxiterations
    new_x = (max_sum - min_sum) * rand + min_sum + x;
    new_y = (max_sum - min_sum) * rand + min_sum + y;
```

5. Dentro do ciclo foram revistos os limites da função, onde caso excedidos, as novas coordenadas passaram a ser as coordenadas limite.

```
% Check Limits

if new_x > x_max
    new_x = x_max;
end
if new_y > y_max
    new_y = y_max;
end
if new_x < x_min
    new_x = x_min;
end
if new_y < y_min
    new_y = y_min;
end
```

6. Verificou-se se a nova solução era “melhor” que a anterior, e em caso afirmativo esta foi marcada no gráfico como um ponto de cor azul.

```
% Does The Change Give a Better Solution?

if ftxy(new_x, new_y) > ftxy(x, y)
    x = new_x;
    y = new_y;

    plot(x, y, 'b.')
end
```

7. Caso esta solução obtida fora então “melhor” que a declarada como tal, passou então essa a apresentar o valor da nova solução.

```
% Check Best for Every Iteration

if ftxy(best_x, best_y) < ftxy(new_x, new_y)
    best_x = new_x;
    best_y = new_y;
end
```

8. Foram adicionadas as coordenadas obtidas aos vetores respetivos. É terminado o “ciclo *for*”.

```
    x_coordenates(k) = x;
    y_coordenates(k) = y;

    txy(k) = ftxy(x_coordenates(k), y_coordenates(k));
end
```

9. Foram criados os gráficos indicadores da atividade: **x** e **y** a cada iteração, ponto por iteração.

```
% Graphics and Representations

figure

title('x and y by Iteration')

plot(x_coordenates, 'b')

hold on

plot(y_coordenates, 'g')

hold off

xlabel('Iterations');
ylabel('x - Blue, y - Green');

%

figure

plot(txy, 'r')
title('')
xlabel('Number of Iterations')
ylabel('ftxy(x, y)')
```

10. Foram obtidos os “melhores” valores para **x** e **y**.

```
%

fprintf('x -> %f\n', best_x)
fprintf('y -> %f\n', best_y)
```

➢ Simulated Annealing

1. Em primeiro lugar, foram definidas o número de iterações, valor inicial da temperatura e iterações por valor de temperatura (*increment*), tal como o fator descendente da mesma e "novo" e "antigo" resultado, para efeitos de comparação.

```
% Initialize Values and Variables

maxiterations = 1000;

temperature = 90;

increment = 10; % iterations per temperature value, second "while"

alpha = 0.94; % lowering temperature factor

old_ftxy = 0;
new_ftxy = 0;
```

2. Foi calculado um ponto inicial aleatoriamente. Tal foi marcado no gráfico da função como um asterisco vermelho.

```
% Random Coordenates

x = (x_max - x_min)*rand + x_min;
y = (y_max - y_min)*rand + y_min;

hold on
plot(x, y, 'r*')
```

3. Foram definidos ciclos "externo" e "interno", correspondentes aos valores de iterações e iterações por valores da temperatura ("primeiro e segundo *while*").

```
% Internal and External Cycles

externalcycle = 0;
internalcycle = 0;
```

4. Inicializados vetores correspondentes à probabilidade e temperatura do ciclo "externo".

```
%

external_temperature = zeros(maxiterations, 1);

external_probability = zeros(increment ,1);
```

5. Inicializados vetores correspondentes à probabilidade associada ao ciclo "interno" tal como a diferença entre o novo e antigo valor.

```
% ftxy(x+1) - ftxy(x)

internal_probability = zeros(increment*maxiterations, 1);
difference = zeros(increment*maxiterations, 1);
```

6. Inicializado vetor correspondente ao ponto obtido, tal como atribuição da primeira posição ao ponto já calculado.

```
% Function Value by Iterations

ftxy_by_iterations = zeros(increment*maxiterations, 1);

% First x and y

ftxy_by_iterations(1) = ftxy(x, y);
```

7. Atribuição da temperatura inicial à primeira posição do vetor temperatura.

```
% Temperature Initialization

external_temperature(1) = temperature;
```

8. Definidos os valores “ótimos” iniciais sendo estes o do ponto inicialmente calculado, para comparação.

```
%

best_x = x;
best_y = y;
```

9. K como número atual de iterações e “Primeiro *while*” a ocorrer até ao máximo número de iterações.

```
%

k = 1; % Iterations Count

while externalcycle < maxiterations

    internalcycle = 1;
```

10. “Segundo *While*” a ocorrer para cada *increment* (iterações por valor de temperatura). Foram também calculados novos **x** e **y**.

```
   while(internalcycle <= increment)
       new_x = (max_sum - min_sum)*rand + min_sum + x;
       new_y = (max_sum - min_sum)*rand + min_sum + y;
```

11. Dentro do ciclo foram revistos os limites da função, onde caso excedidos, as novas coordenadas passaram a ser as coordenadas limite.

```
% Check Limits

if new_x > x_max
    new_x = x_max;
end
if new_y > y_max
    new_y = y_max;
end
if new_x < x_min
    new_x = x_min;
end
if new_y < y_min
    new_y = y_min;
end
```

12. Foram atribuídos os valores de resultado antigo e novo resultado.

```
    old_ftxy = ftxy(x, y);
    new_ftxy =  ftxy(new_x, new_y);
```

13. Caso esta solução obtida fora então “melhor” que a declarada como tal, passou então essa a apresentar o valor da nova solução.

```
% Check Best for Every Iteration

if(ftxy(best_x, best_y) < new_ftxy)
    best_x = new_x;
    best_y = new_y;
end
```

14. Calculou-se a diferença entre os resultados e calculada a probabilidade associada a essa diferença e à temperatura.

```
difference(k) = new_ftxy - old_ftxy;

internal_probability(k) = ProbabilityFunction(difference(k), temperature);
```

Função Probabilidade:

```
function Probability = ProbabilityFunction(EnergyVariation, Temperature)

Probability = exp(-abs(EnergyVariation)./Temperature);

end
```

15. Guardado o valor obtido na posição correspondente.

```
  %

  ftxy_by_iterations(k) = new_ftxy;
```

16. Caso de diferença com valor positivo, consequentemente "melhor" resultado, foram associados os novos **x** e **y**.

```
if difference(k) > 0
    x = new_x;
    y = new_y;
end
```

17. Embora obtenção de solução inferior, tornou-se possível a aceitação desse novo resultado, com base numa probabilidade anteriormente descrita.

```
if rand() < internal_probability(k)
    x = new_x;
    y = new_y;
end
```

18. Acrescentou-se ao ciclo ”interno” e ao número atual de iterações. Foi desenhado o novo ponto de cor azul no gráfico. Término do “segundo *while*” (segundo ciclo).

```
    internalcycle = internalcycle + 1; % Internal Cycle

    plot(x, y, 'b.')

    k = k + 1;
end
```

19. Foi diminuída a temperatura com recurso a um fator *alpha* (de valor indicado anteriormente, 0.94). Acrescentou-se ao ciclo “externo” e guardado o novo valor da temperatura no vetor apropriado de modo a desenhar um gráfico pertinente. Término do “primeiro *While*” (primeiro ciclo).

```
    %

    temperature = temperature*alpha;

    externalcycle = externalcycle + 1;

    external_temperature(externalcycle + 1) = temperature;
end
```

20. Foram criados os gráficos indicadores da atividade: probabilidade a cada iteração, temperatura por iteração.

```
% Graphs

%

figure
plot(internal_probability, 'b')
title('Probability Per Iteration')
xlabel('Iterations')
ylabel('Probability')
%

figure
stairs(external_temperature, 'r') % plot(external_temperature, 'r')
title('Temperature Per Iterations')
xlabel('Iterations')
ylabel('Temperature')
```

21. Foram obtidos os “melhores” valores para **x** e **y**.

```
%

fprintf('x -> %f\n', best_x)
fprintf('y -> %f\n', best_y)
```

➢ Multiple Restart Hill Climbing

1. Após introdução de uma nova variável, *stable*, foi acrescentada a essa nova variável 1 valor em caso de solução “pior” que a anterior.

```
stable = 0;

  % Does The Change Give a Better Solution?

  if ftxy(new_x, new_y) > ftxy(x, y)
      x = new_x;
      y = new_y;

      plot(x, y, 'b.')

      stable = 0;

  else
      stable = stable + 1;
  end
```

2. Caso a variável *stable* tenha apresentado valor 100, foi gerado um novo ponto aleatoriamente e marcado no gráfico como tal.

```
  if stable == 100
      x = (x_max - x_min)*rand + x_min;
      y = (y_max - y_min)*rand + y_min;

      hold on
      plot(x, y, 'r*')

      stable = 0;
  end
```

➢ Multiple Restart Simulated Annealing

1. Após introdução de uma nova variável, *stable*, em caso de diferença com valor negativo, consequentemente "pior" resultado, foi adicionado 1 a essa nova variável.

```
stable = 0;

if difference(k) > 0
    x = new_x;
    y = new_y;
    stable = 0;
else
    stable = stable + 1;
end
```

2. Caso a variável *stable* tenha apresentado valor 100, foi gerado um novo ponto aleatoriamente e marcado no gráfico como tal.

```
 if stable == 100
     x = (x_max - x_min)*rand + x_min;
     y = (y_max - y_min)*rand + y_min;

     plot(x, y, 'r*')
 else
     plot(x, y, 'b.')
 end
```

Comparações

“Simulated-annealing is believed to be a modification or an advanced version of hill-climbing methods.”

Foram analisados os diferentes algoritmos para:

$$a\left(1-x^2\right)e^{-x^2-(1+y)^2}-b\left(\frac{x}{5}-x^3-x^5\right)e^{-x^2-y^2}+\left(-\frac{1}{3}\right)e^{-(x+1)^2-y^2}$$

, a = 3 e b = 10

Em todos os algoritmos foram efetuadas 1000 iterações.

➢ Hill Climbing

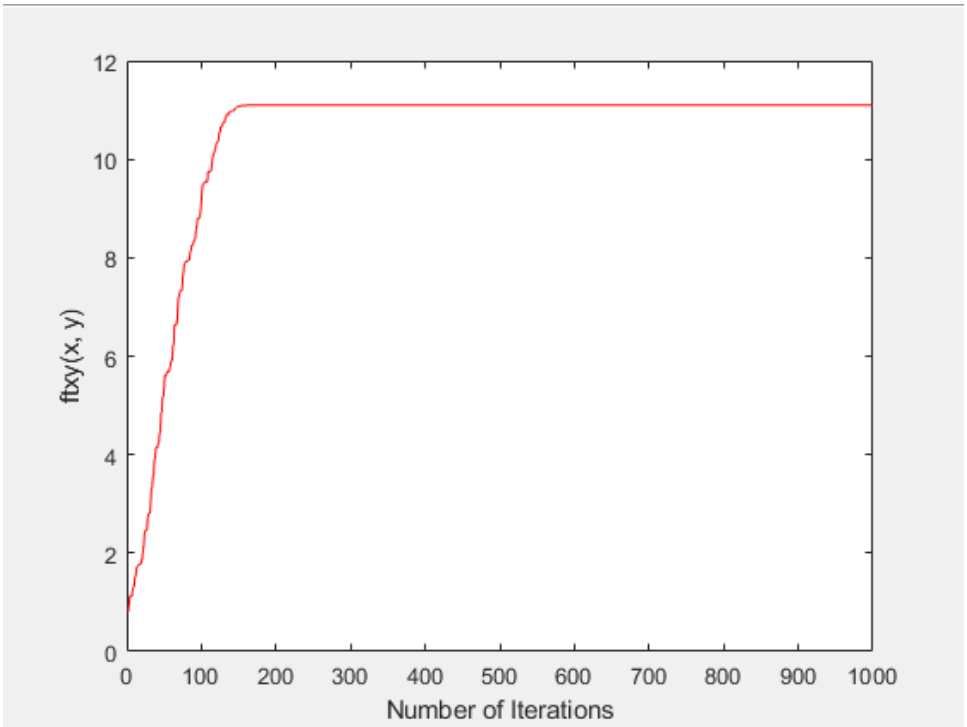

Gráfico 1 - *Hill Climbing*, **f(x,y)** por iteração.

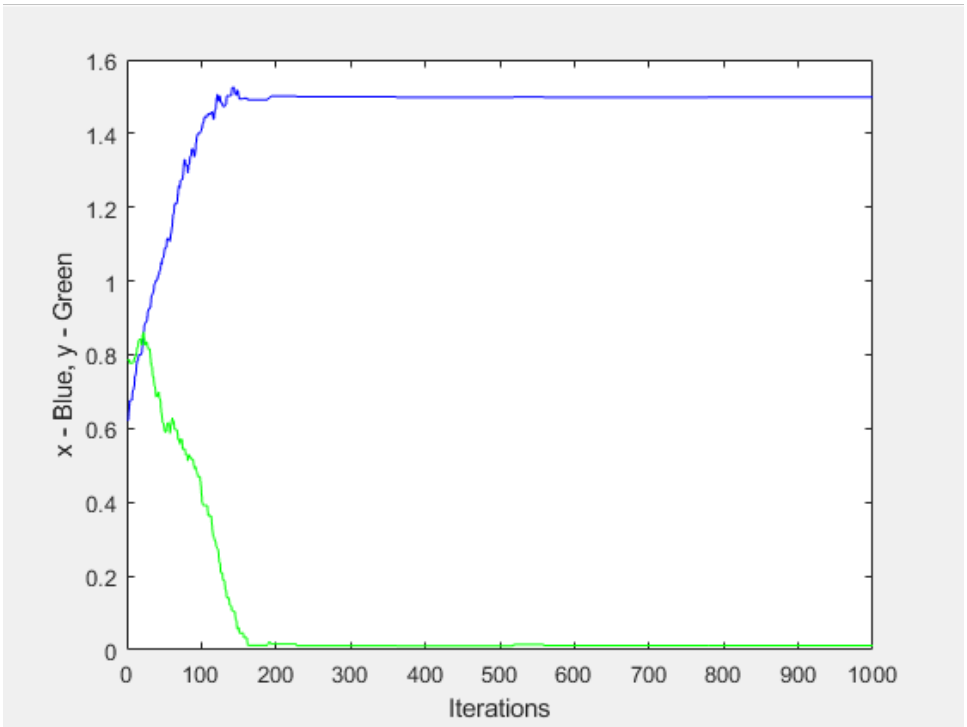

Gráfico 2 - *Hill Climbing*, **x** e **y** por iteração.

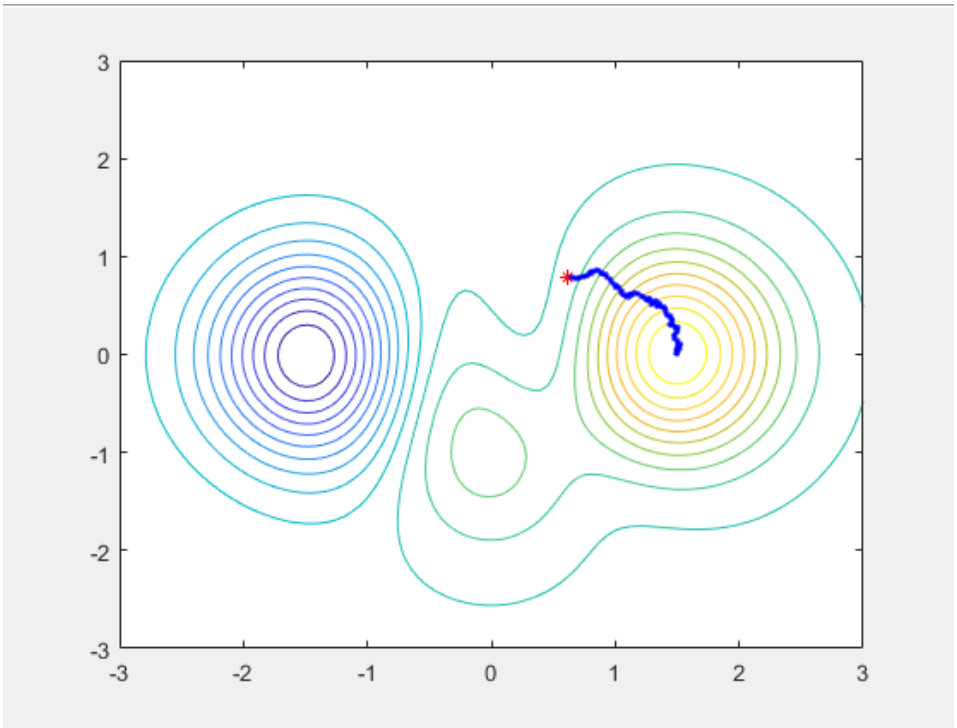

Gráfico 3 - *Hill Climbing*, gráfico total.

```
>> HillClimbing
x -> 1.498771
y -> 0.011789
```

Imagem 1 - *Multiple Restart Hill Climbing*, resultado final.

➢ Multiple Restart Hill Climbing

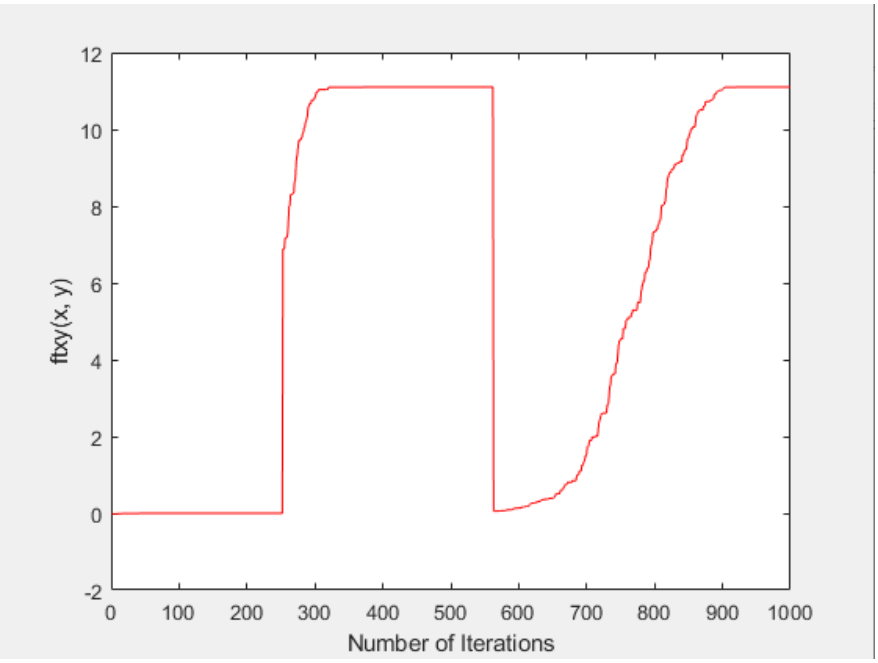

Gráfico 4 - *Multiple Restart Hill Climbing*, **f(x,y)** por iteração.

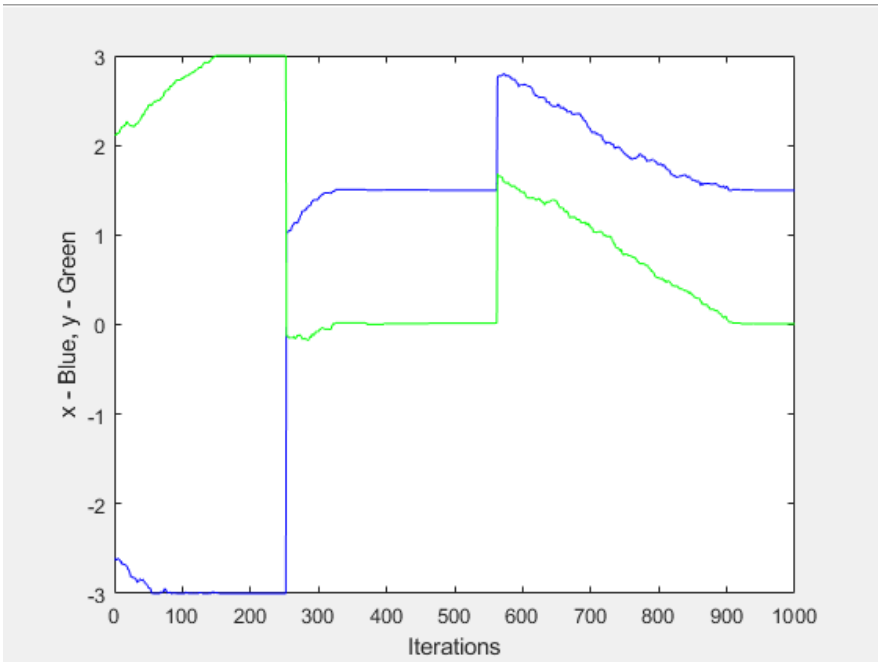

Gráfico 5 - *Multiple Restart Hill Climbing*, **x** e **y** por iteração.

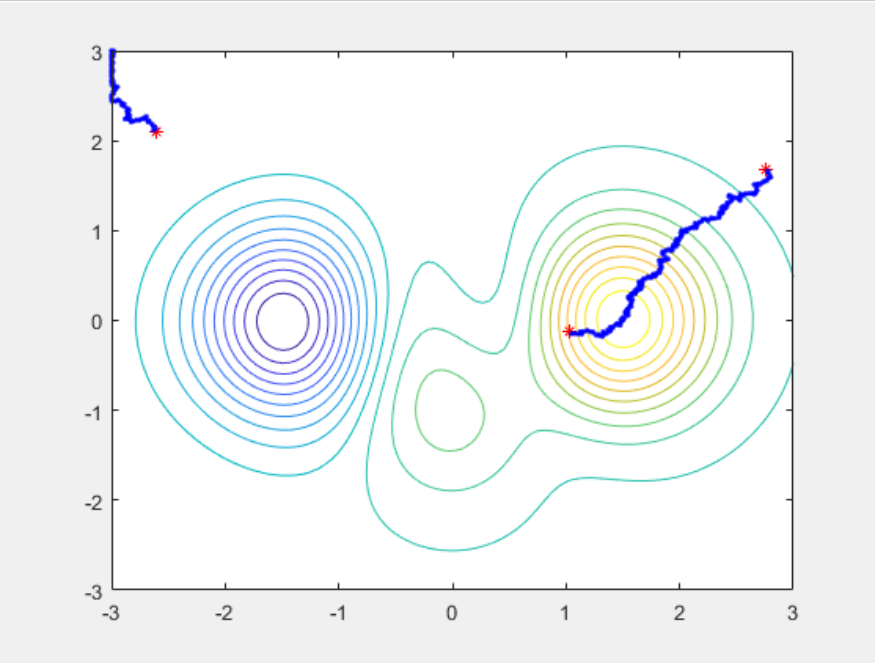

Gráfico 6 - *Multiple Restart Hill Climbing*, gráfico total.

```
>> MultipleRestartHillClimbing
x -> 1.498456
y -> 0.014584
```

Imagem 2 - *Multiple Restart Hill Climbing*, resultado final.

➢ Simulated Annealing

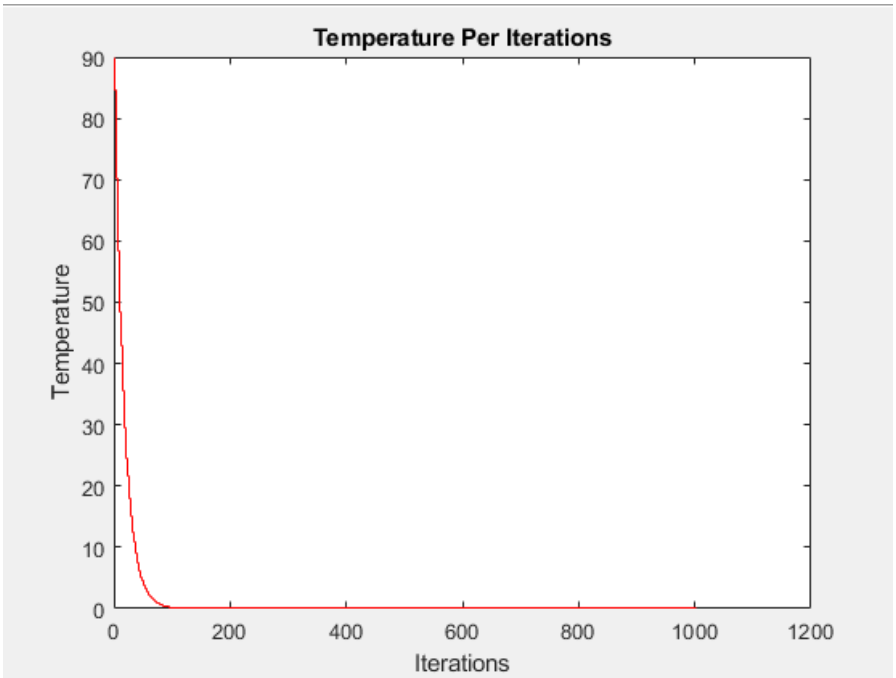

Gráfico 7 - *Simulated Annealing*, temperatura por iteração.

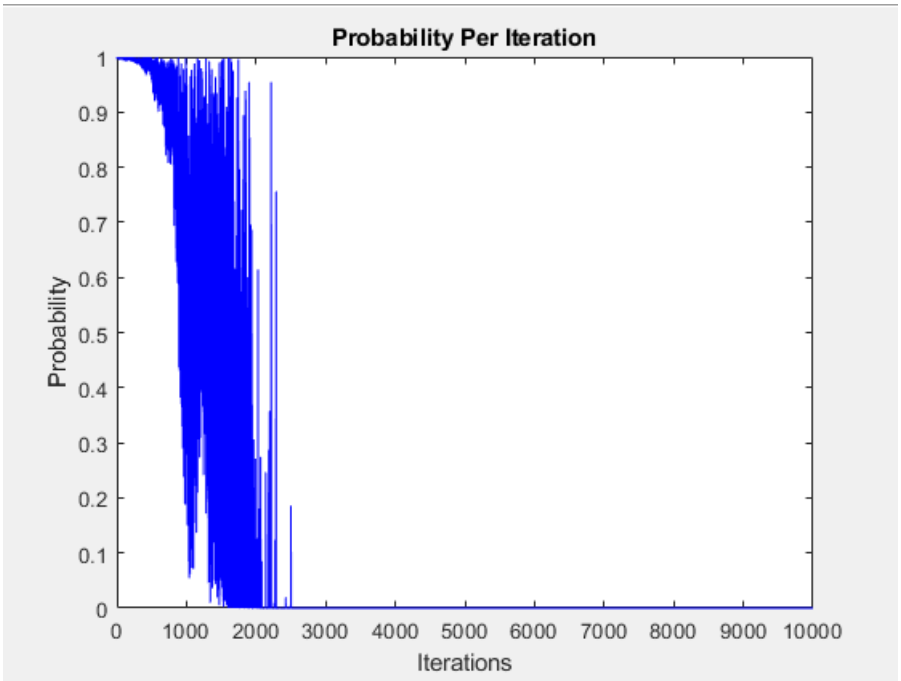

Gráfico 8 - *Simulated Annealing*, probabilidade por iteração.

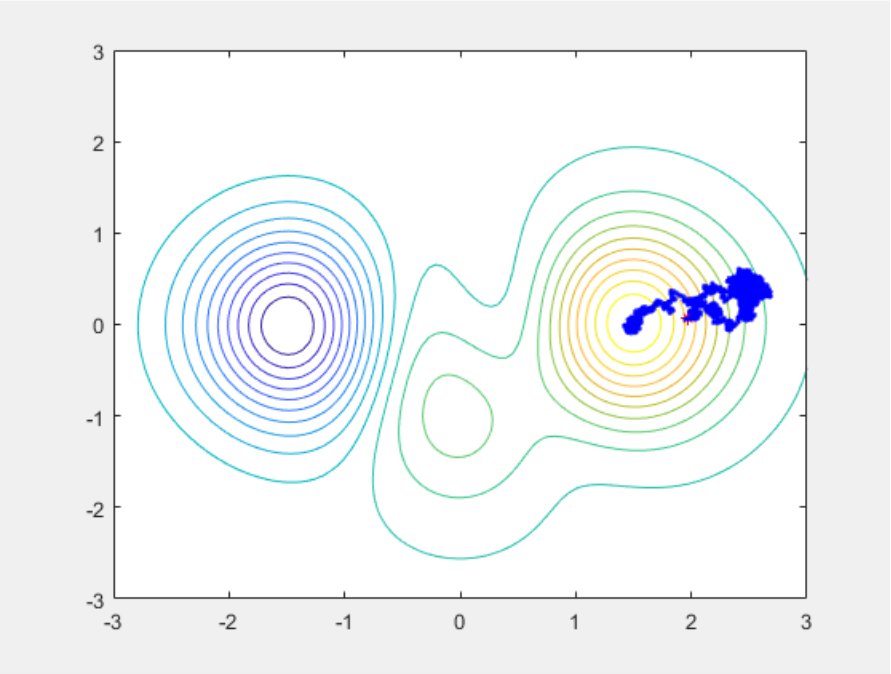

Gráfico 9 - *Simulated Annealing*, gráfico total.

```
>> SimulatedAnnealing
x -> 1.498493
y -> 0.013017
```

Imagem 3 - *Simulated Annealing*, resultado final.

➢ Multiple Restart Simulated Annealing

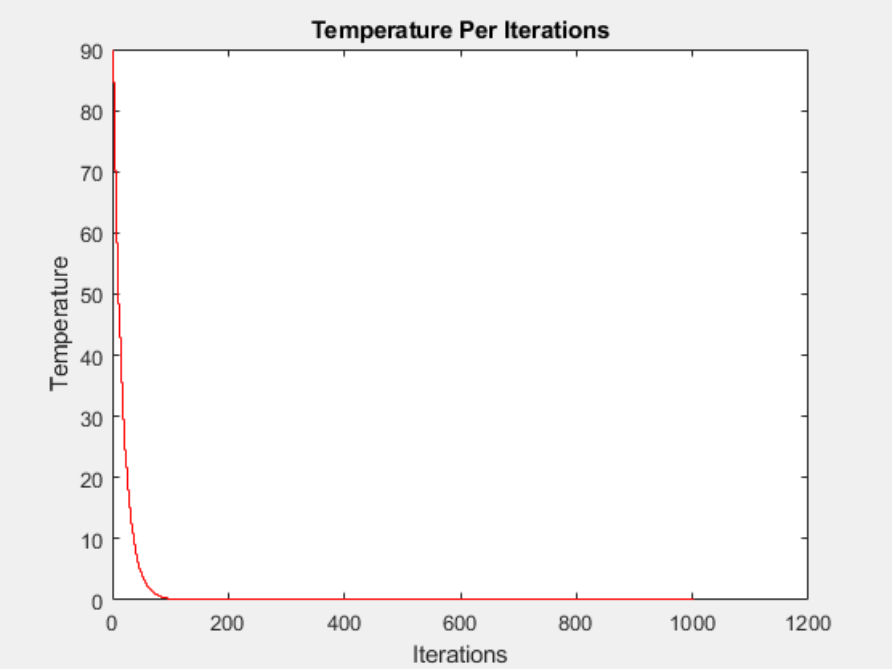

Gráfico 10 - *Multiple Restart Simulated Annealing*, temperatura por iteração.

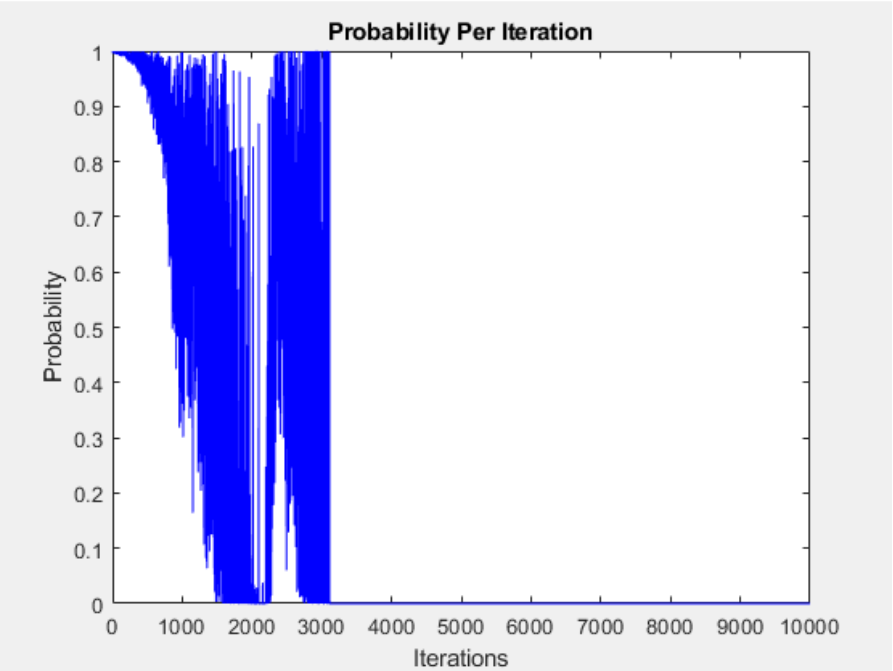

Gráfico 11 - *Multiple Restart Simulated Annealing*, probabilidade por iteração.

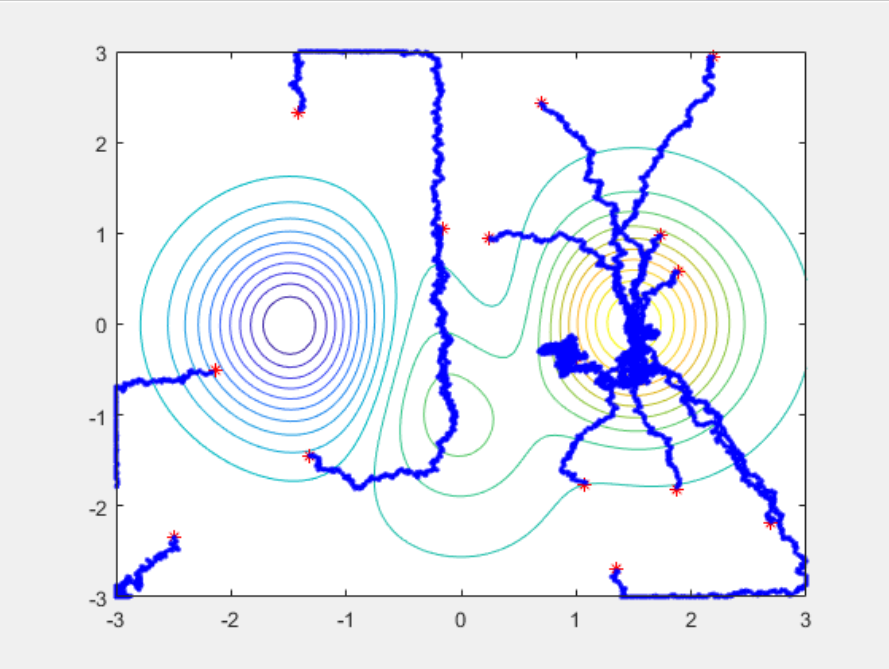

Gráfico 9 - *Multiple Restart Simulated Annealing*, gráfico total.

```
>> MultipleRestartSimulatedAnnealing
x -> 1.498742
y -> 0.012617
```

Imagem 4 - *Multiple Restart Simulated Annealing*, resultado final.

➢ Multiple Restart Simulated Annealing, a = 9 e b = 5

Foi analisado o presenta algoritmo para:

$$a\left(1-x^2\right)e^{-x^2-(1+y)^2}-b\left(\frac{x}{5}-x^3-x^5\right)e^{-x^2-y^2}+\left(-\frac{1}{3}\right)e^{-(x+1)^2-y^2}$$

, a = 9 e b = 5

1. Modificou-se o ciclo externo de modo a quando são efetuadas metade das iterações definidas, a = 9 e b = 5.

```
while externalcycle < maxiterations

    if k >= maxiterations
        a = 9;
        b = 5;
    end
```

Foram obtidos os seguintes resultados:

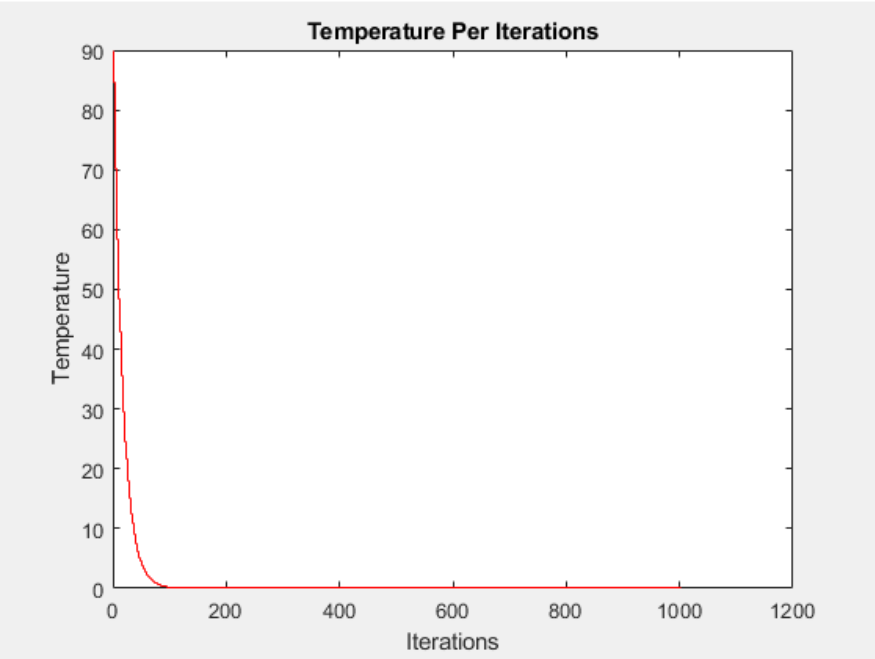

Gráfico 11 - *Multiple Restart Simulated Annealing Change*, temperatura por iteração.

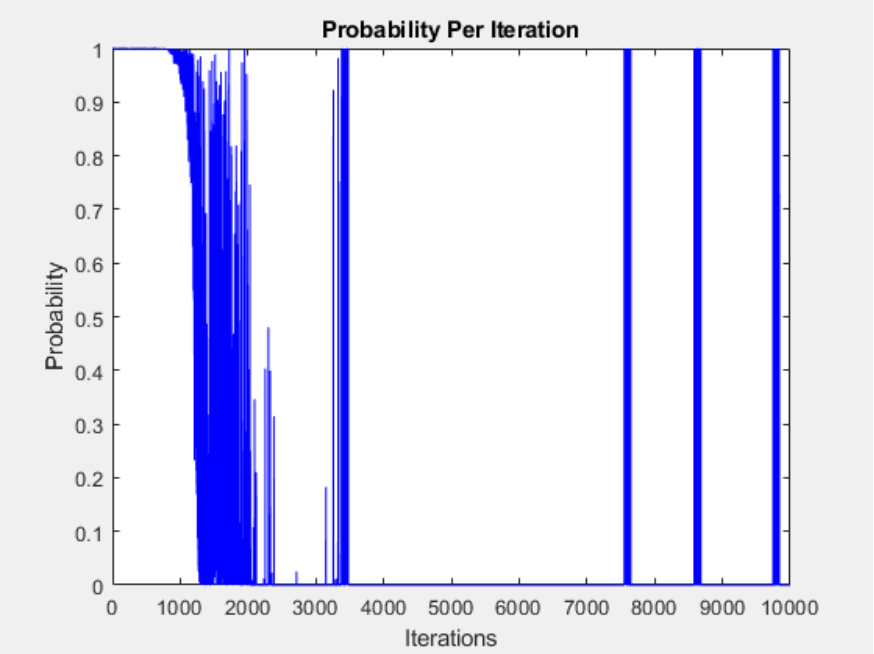

Gráfico 11 - *Multiple Restart Simulated Annealing Change*, probabilidade por iteração.

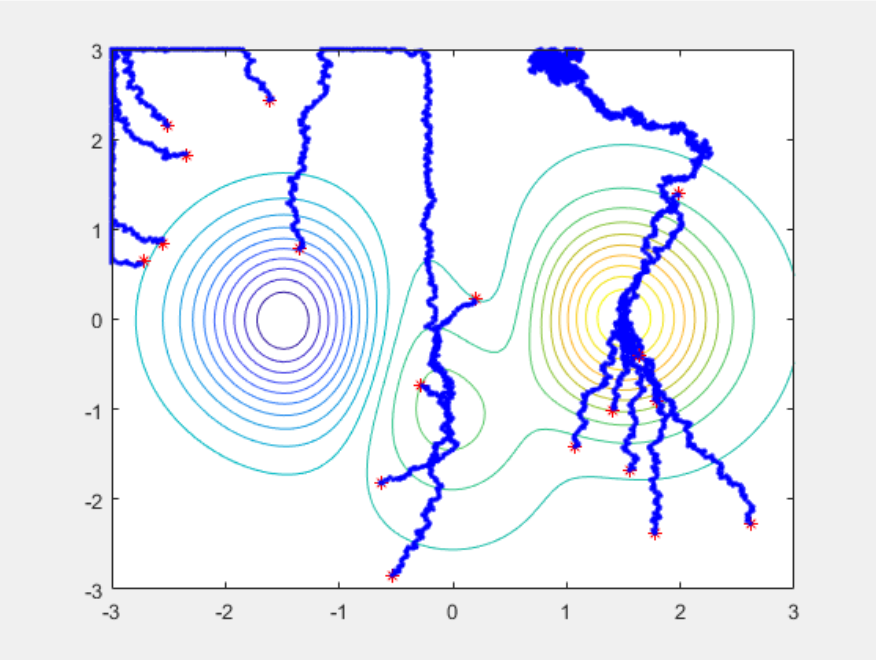

Gráfico 12 - *Multiple Restart Simulated Annealing Change*, gráfico total.

```
>> MultipleRestartSimulatedAnnealingChange
x -> 1.497830
y -> 0.012301
```

Imagem 5 - *Multiple Restart Simulated Annealing Change*, resultado final.

➢ Random Walk

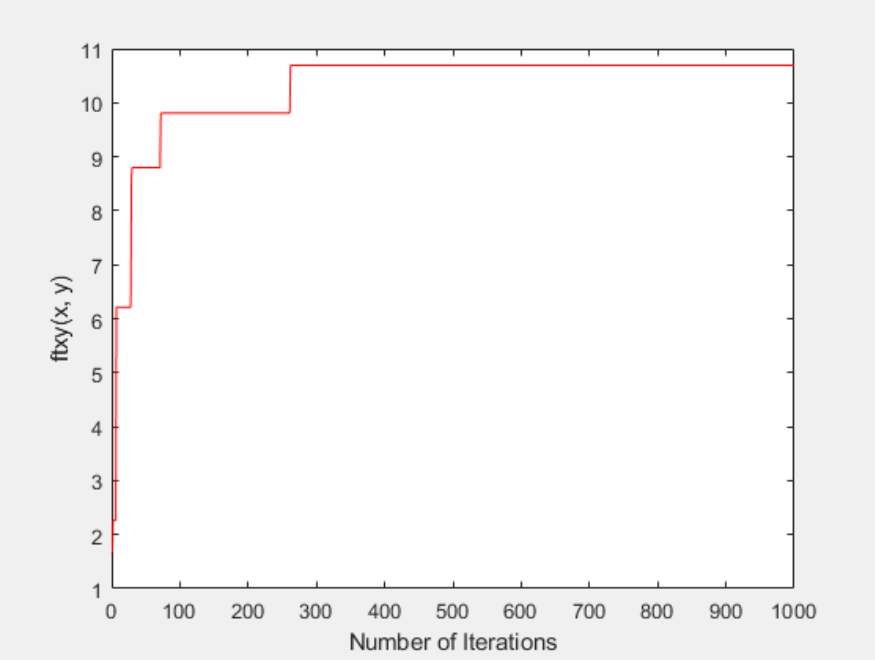

Gráfico 13 - *Random Walk*, **f(x,y)** por iteração.

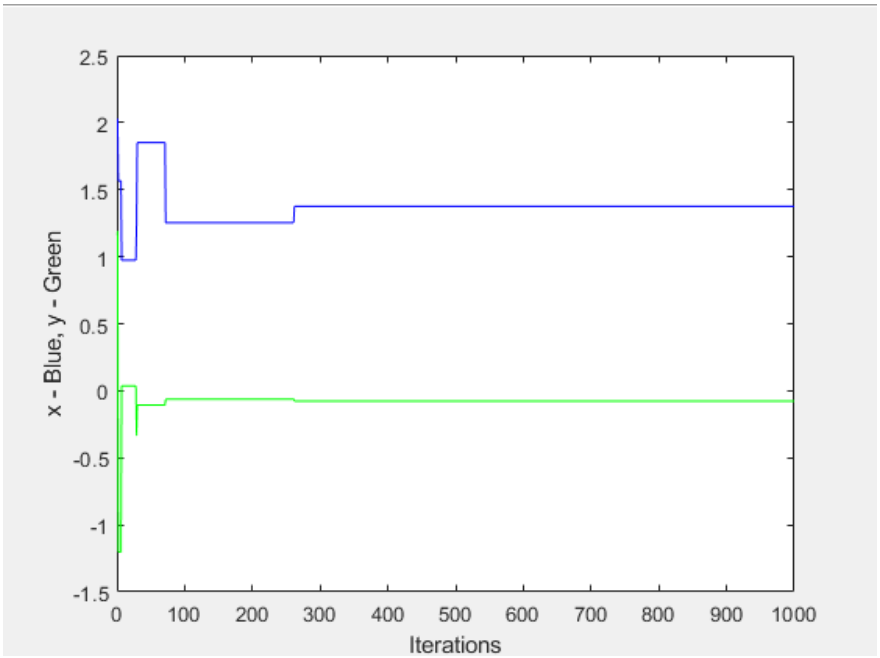

Gráfico 14 - *Random Walk*, **x** e **y** por iteração.

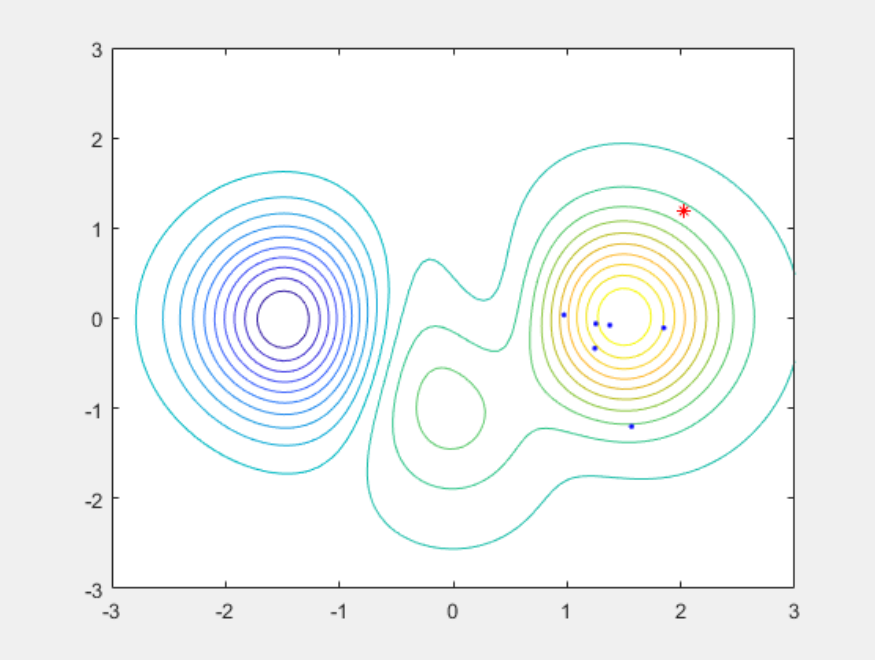

Gráfico 15 - *Random Walk*, gráfico total.

```
>> RandomWalk
x -> 1.375751
y -> -0.077958
```

Imagem 6 - *Random Walk*, resultado final.

Observações

Foram obtidas soluções aproximadas de “ótimos globais”.

Por vezes foram obtidos “ótimos locais”, por parte dos métodos sem reinicialização.

Conclusão

A utilização de ferramentas de programação resulta em soluções nas quais dificilmente chegaríamos utilizando outros meios, ou ainda, gastaríamos tanto tempo que seria inviável.

Contudo, a utilização de algoritmos na pesquisa não é a melhor prática.

Algoritmos como *Hill Climbing* podem até apresentar uma versão muito aproximada dos resultados ótimos locais, no entanto, não é exata.

Simulated Annealing recorre também à aproximação de modo a encontrar um ótimo global, desfazendo-se em tentativas que resultam apenas numa aproximação.

O mesmo sucede aquando da utilização da reinicialização nestes algoritmos. São de tentativa e erro, não são precisos e baseiam-se apenas na capacidade de validar o maior número de indivíduos possível.

Para obterem valores exatos teriam de ser calculados todos os pontos do universo em questão, sendo impossível, dado que os números são infinitos.

Estes algoritmos são vantajosos quando é pretendida uma aproximação da solução.

Referências Bibliográficas

Russell, S. J., Russell, S. J., Norvig, P., & Davis, E. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Lucci, S., & Kopec, D. (2016). *Artificial Intelligence in the 21st Century*. Mercury Learning and Information.

Anexos

[1] Procedimento de alto nível designado para encontrar, gerar, ou selecionar uma heurística que pode proporcionar uma solução suficiente.