UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

Trabalho Prático 2

Relatório do trabalho prático sobre Agentes Inteligentes

Engenharia Informática

Inteligência Artificial

Autores

Guilherme Cruz – al73752

Tiago Fernandes - al73701

Vila Real, 2022

Índice

1. Introdução 4
2. Hill Climb – Método da Subida da Colina 5
Introdução ao *Hill Climbing* 5
Implementação do método *Hill Climbing* no Problema 6
3. Simulated Annealing 11
Introdução ao *Simulated Annealing* 11
Implementação do método Simulated Annealing 11
4. Conclusão 16
5. Referências 17
6. Anexos 18
Hill Climbing com reinicialização 18
Simulated Annealing com 2 funções 20

1. Introdução

No contexto da unidade curricular de inteligência artificial, foi pedido a entrega de um relatório sobre os Algoritmos lecionados, estes sendo: o algoritmo Hill-Climb (com e sem reinicialização) e o algoritmo de Simulated Annealing. Neste relatório estão contidas implementações dos algoritmos mencionados, sendo estes feitos na aplicação MatLab. Este Trabalho está divido em 2 partes, sendo a primeira uma introdução teórica aos algoritmos expostos. A segunda parte do trabalho contém a implementação pratica do Hill Climb com a sua reinicialização e o algoritmo Simulated Annealing, este no final usado com 2 formulas diferentes.

2. Hill Climb – Método da Subida da Colina

Introdução ao *Hill Climbing*

Hill Climbing é um método de busca heurística usado para problemas de otimização matemática na área de Inteligência Artificial. 'Pesquisa heurística' significa que este algoritmo de busca pode não encontrar a solução ótima para o problema. No entanto, dará uma boa solução em um tempo razoável. Uma função heurística é uma função que classificará todas as alternativas possíveis em qualquer etapa de ramificação no algoritmo de busca com base nas informações disponíveis. Ajuda o algoritmo a selecionar a melhor rota dentre as rotas possíveis.

Dado uma função, ele tenta encontrar uma solução suficientemente boa para o problema. Esta solução pode não ser o máximo ótimo global. Este método poderá ser usado de forma maximizante ou minimizante dependendo do problema existente.

O *Hill Climbing* tem como características, ser uma variante do algoritmo de geração e teste, pois:

1. Gera possíveis soluções;
2. Testa para ver se é o resultado esperado;
3. Se a solução foi encontrada saí, caso contrário volta ao passo 1;

Ou seja, este recebe o *feedback* do procedimento de teste. Então este *feedback* é utilizado pelo gerador para decidir o próximo movimento no espaço de busca O *Hill Climbing* tem como também característica usar um "*Greedy approach*" (abordagem gananciosa) isto porque a busca apenas se move numa direção em que otimize o custo da função, na esperança que no final encontre uma ótima solução.

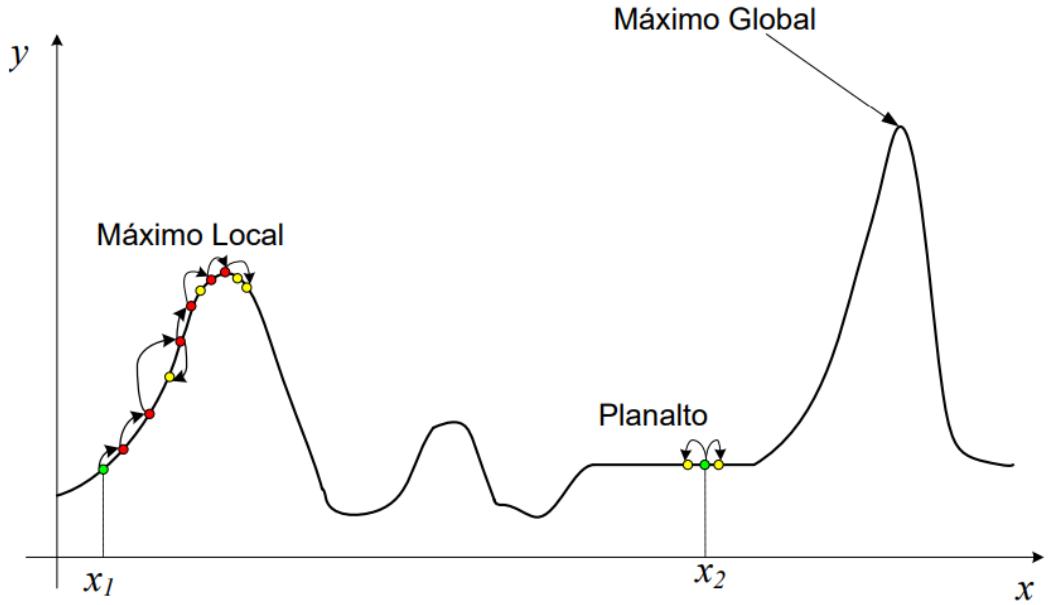


Figura 1- Regiões existentes num gráfico

O *Hill Climbing* acaba por vir a sofrer com alguns problemas, tal como os Planaltos e/ou Planícies e os Máximos Locais. Nos Planaltos e nas Planícies todos os vizinhos têm o mesmo valor. Portanto, não é possível selecionar a melhor direção. Já nos Máximos Locais o que acontece é que todos os estados vizinhos têm um valor pior que o estado atual. Como o *Hill Climbing* usa um "*Greedy approach*" ela não se irá mover para um local pior dando assim como terminado o processo, mesmo existindo solução melhor.

Com o objetivo de colmatar estes problemas foram criadas variantes do *Hill Climbing*, sendo que neste trabalho foi usado o *Multiple Restart Hill Climbing.* Este consiste em que cada vez que chega a uma zona na qual o algoritmo não mexe, existe uma reinicialização do ponto o que poderá fazer com que este tenha novos percursos e aumente a probabilidade chegar ao Máximo Absoluto.

Implementação do método *Hill Climbing* no Problema

Na unidade curricular foi entregue um problema que tinha como objetivo descobrir o máximo global da Função:

```
%Funções
        g1 = @(x,y) 9.*(1-x.^2).*exp(-x.^2 -(1+y).^2);
        g2 = @(x,y) -5*(x/5 - x.^3 - y.^5).*exp(-x.^2 - y.^2);
        g3 = @(x,y) -1/3 * exp(-(x+1).^2 -y.^2);
        %função principal
        gxy = @(x,y) abs(g1(x,y) + g2(x,y) + g3(x,y));
```

Assim com o objetivo de resolver o problema imposto criámos um algoritmo MRHC que reinicializa aquando da estagnação, este dentro de um intervalo de 1000 iterações.

Este algoritmo então cria um ponto aleatório entre -3<x<3 e -3<y<3, em seguida é gerado um novo ponto aleatório na sua vizinhança, se este for um ponto melhor que o primeiro este irá ser guardado como o valor de comparação, se este não for melhor que o ponto existente será usado o ponto existente como valor de comparação. Isto repete--se enquanto não existir uma visível estagnação, esta dando se quando o ponto se encontra igual no espaço de 10 iterações. Existindo uma estagnação a função volta a gerar um novo ponto, após gerar novo ponto esta volta entrar no mesmo ciclo até um novo ponto de estagnação, a função acabada de forma definitiva às 1000 iterações.

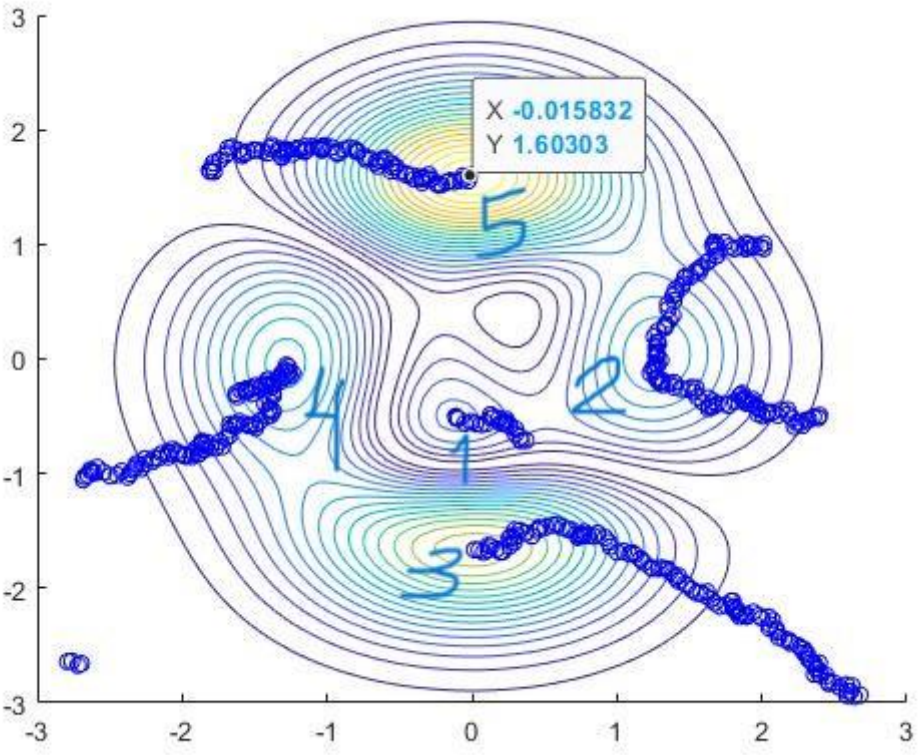


Figura 2 - Gráfico da função e das rotas dos pontos

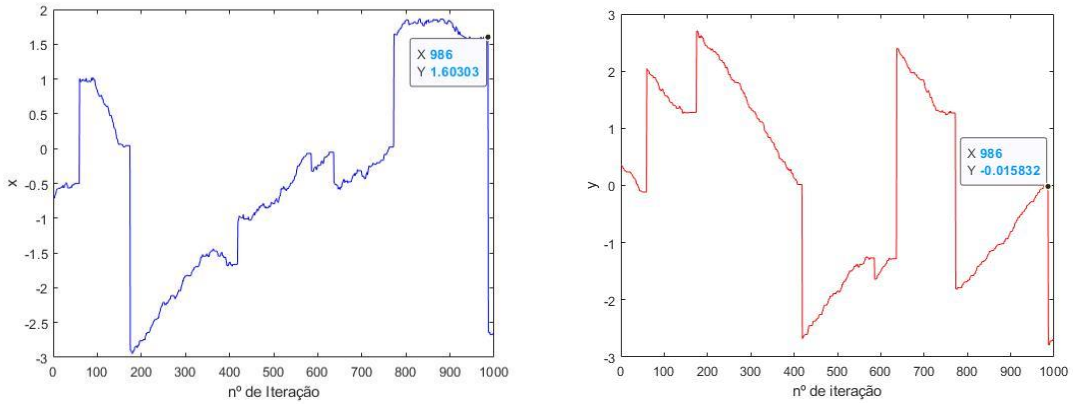


Figura 3 e 4 - Gráfico da evolução de x e y

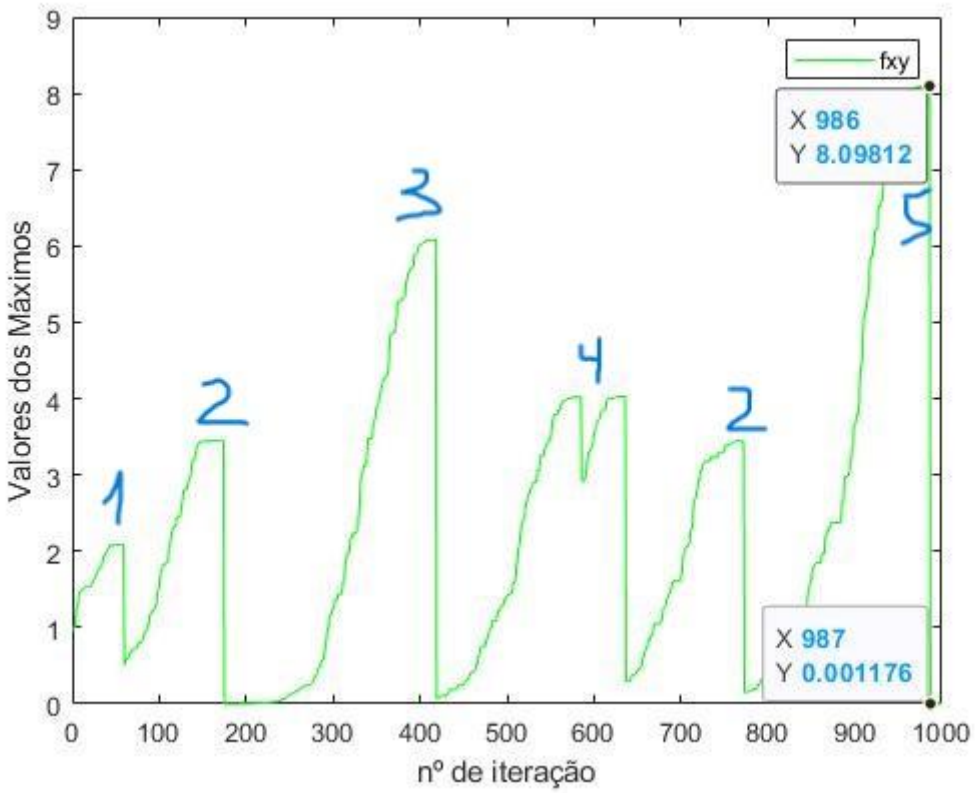


Figura 5 - Gráfico da Evolução do valor do Máximo

Podemos verificar ao analisar as imagens que um valor próximo do Máximo Global foi encontrado no ponto (-0.015832, 1.60303) marcado como nº 5 nas figuras 1 e 4. Como podemos verificar na figura 4, é visível que apesar deste encontrar o máximo global, não significa que a função pare nesse momento pois a função não tem maneira de descobrir se a função é verdadeiramente o máximo global. Assim como referido antes a função não tem como saber qual é o ponto máximo, assim iremos encontrar exemplos de testes nos quais não é encontrado o máximo global, como é o exemplo:

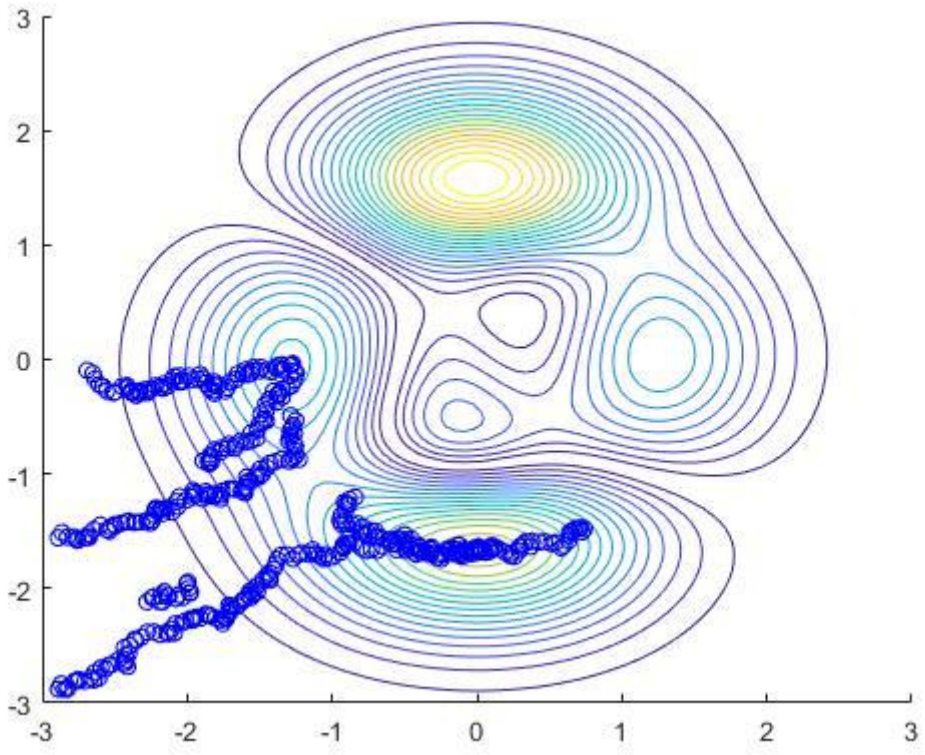
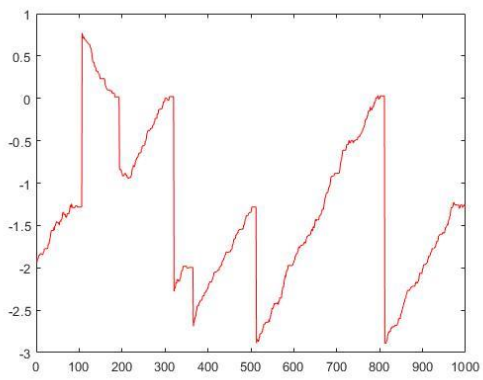


Figura 6 - Gráfico da função e das rotas dos pontos



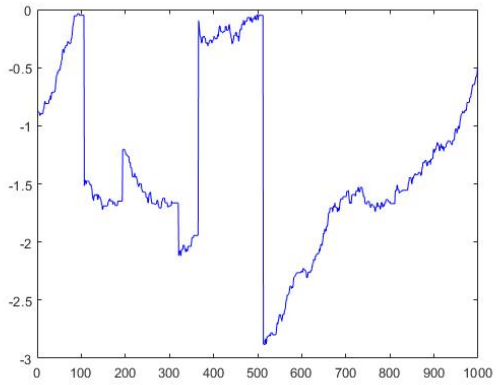


Figura 7 e 8 - Gráfico da evolução de x e y

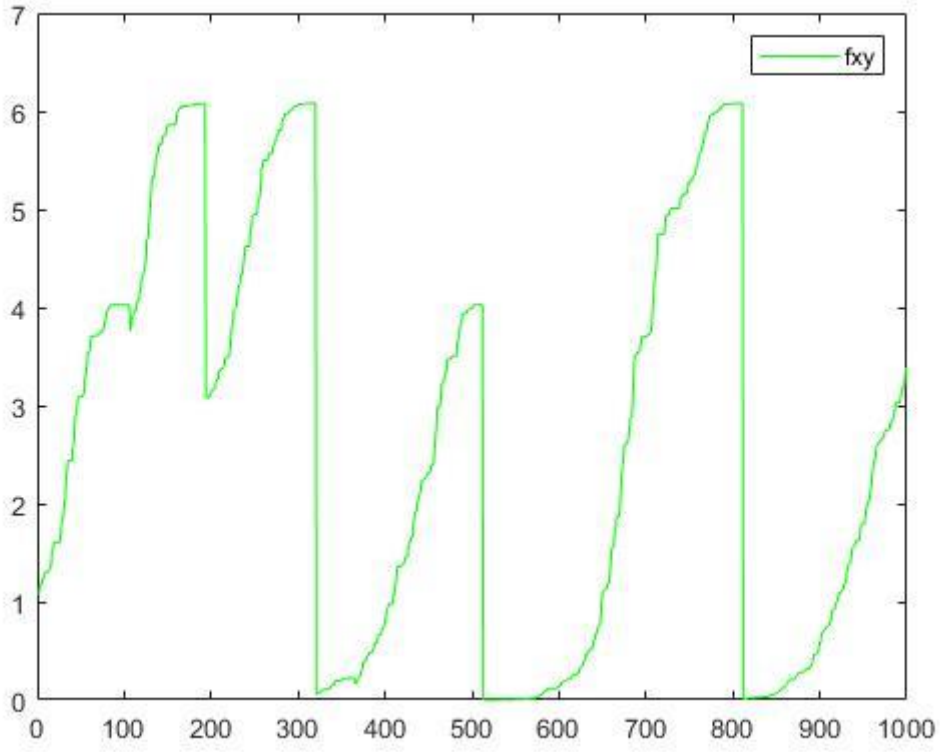


Figura 9 - Gráfico da Evolução do valor do Máximo

Olhando para este teste conseguimos ver que várias soluções locais foram encontradas, no entanto, e apesar de haver 5 reinicializações de pesquisa, a solução global não foi encontrada.

Deste modo concluímos que o nosso algoritmo em relação ao algoritmo clássico (Hill Climbing) foi otimizado sendo possível encontrar a solução desejada.

3. Simulated Annealing

Introdução ao *Simulated Annealing*

Simulated Annealing (SA) é uma forma eficaz e geral de otimização. É útil para encontrar ótimos globais na presença de um grande número de ótimos locais. "*Annealing*" refere-se a uma analogia com a termodinâmica, especificamente com a maneira como os metais arrefecem e recozem. O *Simulated Annealing* usa a função objetivo de um problema de otimização em vez da energia de um material.

A implementação do *Simulated Annealing* é surpreendentemente simples. O algoritmo é como uma escalada, exceto que, em vez de escolher o melhor movimento, e sempre subir a montanha, ele escolhe um movimento aleatório. Se o movimento selecionado melhorar a solução, ele será sempre aceito. Caso contrário, o algoritmo faz o movimento de qualquer maneira com alguma probabilidade menor que 1. A probabilidade diminui exponencialmente com a mudança da temperatura e com a piora do movimento, que é a quantidade deltaE pela qual a solução é piorada (ou seja, a energia é aumentada). A probabilidade de este selecionar um ponto aleatório diminui com a diminuição da temperatura. Com estes pontos aleatórios, em comparação a outros métodos como o método *Hill Climbing* este traz as suas vantagens pois com os pontos aleatórios aumenta a probabilidade de superar os problemas de máximos locais ou de planaltos ou planícies.

Apesar de tudo, este método a semelhança com o *Hill Climbing* não reconhece o máximo global, o que poderá vir a aparecer problemas como o ponto gerado aleatório ser implementado no máximo global, mas como este não o reconhece como tal, este executa um novo salto aleatório para um ponto desfavorável, para o problema.

Implementação do método Simulated Annealing

Com o objetivo de resolver o problema imposto, acima mencionado, e com o extra de mudar a função dada, criamos um algoritmo *Simulated Annealing* na qual muda a função aquando da estagnação, quando volta a existir a estagnação da segunda função o programa termina, mostrando o local de estagnação da função, este que no programa é chamado de máximo_func1 e máximo_func2.

Este algoritmo então cria um ponto aleatório entre -3<x<3 e -3<y<3, em seguida é gerado um novo ponto aleatório na sua vizinhança se este for um ponto melhor que o primeiro este irá ser guardado como o valor de comparação, se este não for melhor que o ponto existente será executado um *"rand"* na qual se este for um valor abaixo da

probabilidade, que é o resultado da exponencial de deltaE/Temperatura, o ponto irá ser guardado como o novo ponto, senão é mantido o novo ponto. Esta probabilidade com a evolução do método diminui, devido também á diminuição da temperatura. Isto repete-se enquanto não existir uma visível estagnação, esta dando se quando o ponto se encontra igual no espaço de 10 iterações. Existindo uma estagnação é feita uma troca de função e volta a ser gerado um novo ponto, após gerar novo ponto esta volta entrar no mesmo ciclo até um novo ponto de estagnação, quando esta estagna a segunda vez o método chega ao seu fim.

Após vários testes no programa verificamos que a vizinhança iria influenciar bastante o resultado, pois quando a variável que controlava a aleatoriedade do ponto é baixa é tendencioso que o máximo dado seja um ponto perto do local do ponto inicial, como é o exemplo:

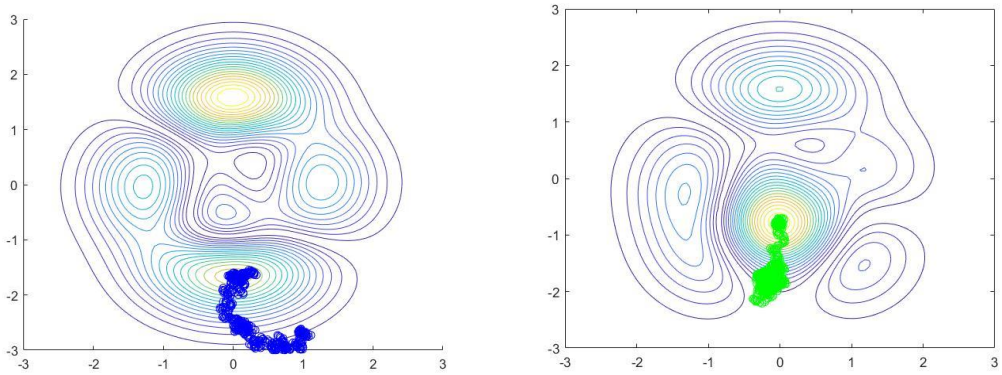


Figura 10 e 11 - Gráficos das funções e rotas 1 e 2 com variância baixa

Então para colmatar este problema usamos uma variável de vizinhança que em conformidade com a Temperatura vai diminuindo, neste caso usando: (1,5/90)*T, aqui 1,5 corresponde á variável máxima da vizinhança, os 90 corresponde a temperatura inicial e T corresponde a temperatura naquela iteração, quando a temperatura atinge 2,5º a variável a partir dai será sempre 0,05.

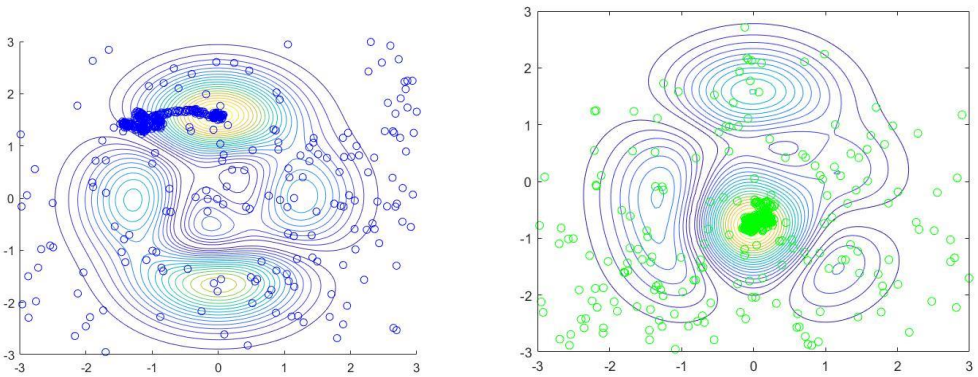
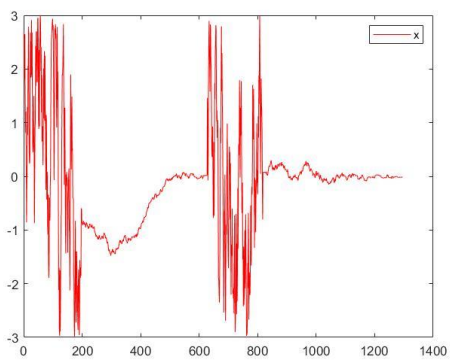


Figura 12 e 13 - Gráficos das funções e das rotas 1 e 2



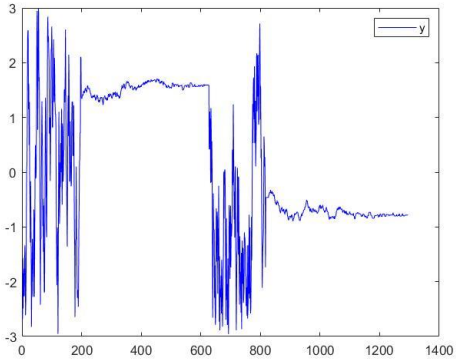


Figura 14 e 15 - Gráficos da evolução do x e y

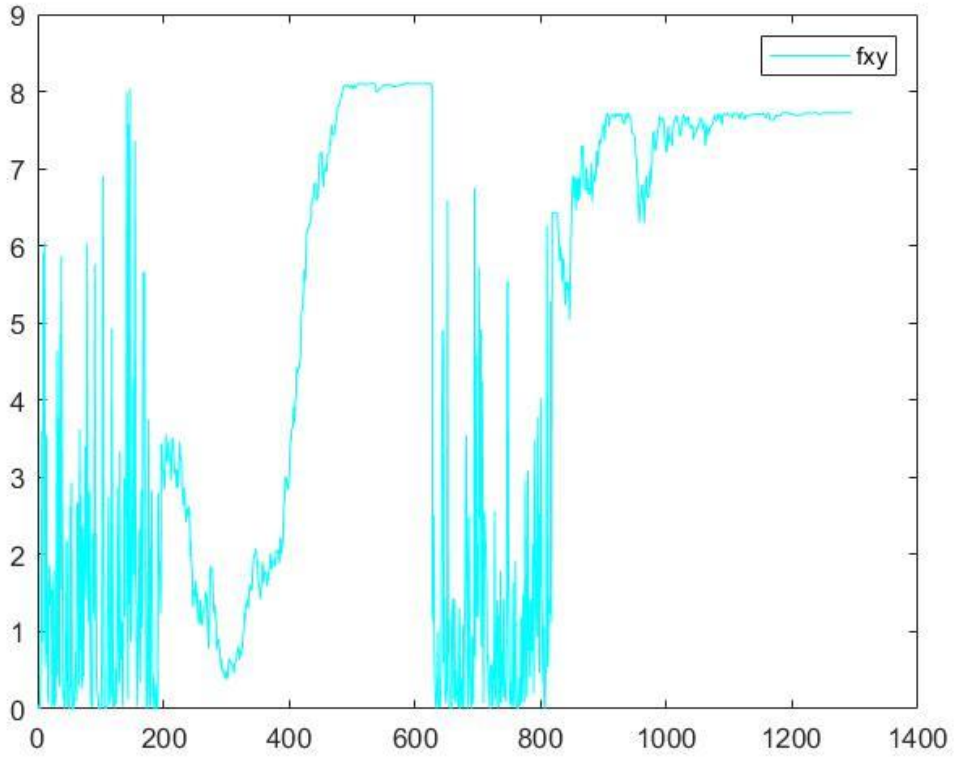
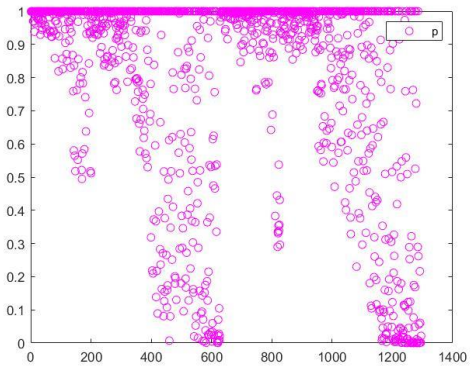


Figura 16 - Gráfico da evolução dos máximos



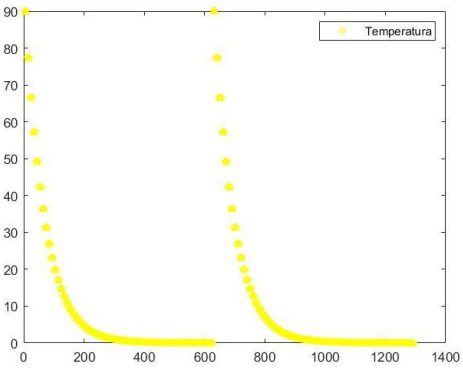
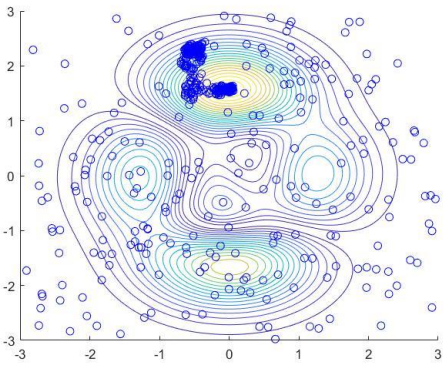


Figura 17 e 18 - Gráficos da evolução da probabilidade e da Temperatura

Apesar deste método com a variável da vizinhança volátil permitir melhores resultados que o com uma variável fixa tanto baixa como alta (esta não usamos de exemplo pois como é percetível no exemplo dela volátil quando a variável é alta faz uma enorme dispersão de dados na qual dificilmente encontraremos um máximo), este não está a salvo de erros e não significa que irá sempre dar o máximo global como é o exemplo a cima, pois quando a temperatura desce e a variável da vizinhança também este poderá estar perto de um máximo relativo e a probabilidade estar baixa irá influenciar com que este suba para o máximo relativo pois não pode dar saltos de pontos grandes, nem é tão provável aceitar pontos “piores”. Este exemplo podemos verificar nas figuras abaixo:



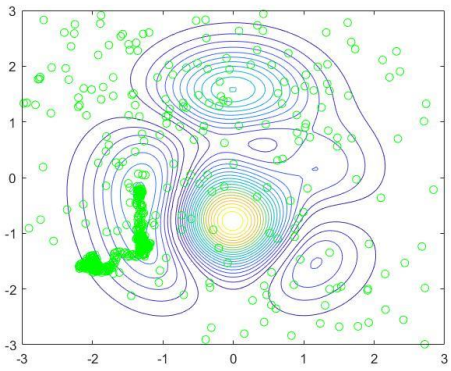


Figura 19 e 20 - Gráficos das funções e rotas 1 e 2

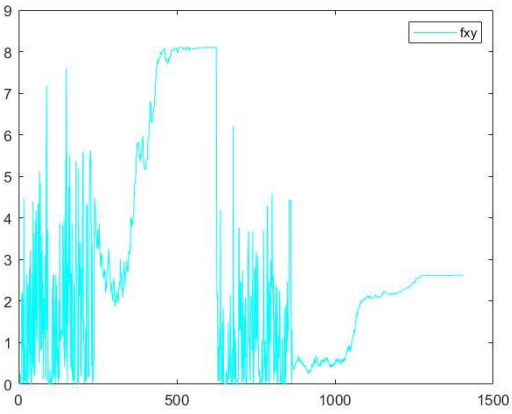


Figura 21 - Gráfico da evolução dos máximos

Podemos assim verificar ao analisar as figuras acima e ao lado um exemplo da execução do programa a qual este não descobriu o máximo global da função 2 e por outro lado a função chegou ao seu máximo global, isto se dá devido a zona em que os pontos se dispersaram aquando da diminuição da variável de vizinhança.

4. Conclusão

Após a resolução deste trabalho prático, chegámos à conclusão de que o objetivo proposto foi atingido com sucesso. Através da utilização da aplicação MatLab, que já tínhamos utilizado em anos anteriores, tivemos a oportunidade de conhecer melhor os algoritmos *Hill-Climbing* e *Simulated Annealing*.

Assim conseguimos concluir que a realização deste trabalho prático nos ajudou a entender mais sobre a implementação de algoritmos de otimização. Nas páginas seguintes temos as referências bibliográficas e os scripts feitos neste trabalho.

5. Referências

- Carneiro, A. L. C. (2020, junho 24). Algoritmos de otimização: Hill Climbing e simulated annealing. *Data Hackers*. https://medium.com/data-hackers/algoritmos-de-otimiza%C3%A7%C3%A3o-hill-climbing-e-simulated-annealing-3803061f66f0
- Colnago, A. C., & Lima, R. H. P. (sem data). *Desenvolvimento de um algoritmo Hill-Climbing para minimização do makespan em Problemas de Sequenciamento em Flow Shops*. 11.
- *Hill Climbing Algorithm in AI - Javatpoint*. (sem data). Www.Javatpoint.Com. Obtido 5 de dezembro de 2022, de https://www.javatpoint.com/hill-climbing-algorithm-in-ai
- Simulated Annealing. (2017, agosto 11). *GeeksforGeeks*. https://www.geeksforgeeks.org/simulated-annealing/
- *Understanding Hill Climbing Algorithm in Artificial Intelligence*. (sem data). Engineering Education (EngEd) Program | Section. Obtido 5 de dezembro de 2022, de https://www.section.io/engineering-education/understanding-hill-climbing-in-ai/

6. Anexos

Hill Climbing com reinicialização

```
%Funções
f1 = @(x,y) 3.*(1-x.^2).*exp(-x.^2 -(1+y).^2);
f2 = @(x,y) -10*(x/5 - x.^3 - y.^5).*exp(-x.^2 - y.^2);
f3 = @(x,y) -1/3 * exp(-(x+1).^2 -y.^2);
%Funçao principal
fxy = @(x,y) abs(f1(x,y) + f2(x,y) + f3(x,y));

%%fxy = @(x,y) 3.*(1-x.^2).*exp(-x.^2 -(1+y).^2);

close all;

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

%Geração de ponto aleatorio
rx=(rand-0.5)*2*3;
ry=(rand-0.5)*2*3;

hold on
%plot(sx,sy,'b*')

%parametros da figura
vx=linspace(-3,3,100);
vy=linspace(-3,3,100);

[X,Y]=meshgrid(vx,vy);


FX=fxy(X,Y);

%desenho da figura
contour(X,Y,FX,20);

%nº de iterações
n=1000;

%vetores
vx = zeros(n,1);
vy = zeros(n,1);
vxy = zeros(n,1);
vmax = zeros(2,1);

aux= 1;

for it = 1:n
d=0.05;%vizinhança

%geração de novos pontos
  new_x= rx + (rand-0.5)*2*d;
  new_y= ry + (rand-0.5)*2*d;
```

```
    %verifica se está mais alto que o guardado
    if fxy(rx,ry)<fxy(new_x,new_y)

        rx=new_x;
        ry=new_y;
        plot(rx,ry,'bo') % colocar pontos no grafico
    end

    %guardar pontos no vetor
    vx(it) = rx;
    vy(it) = ry;
    vxy(it) = fxy(rx,ry);

    %verificar se estabilizou
    if(it>10 && vxy(it-10)==vxy(it))

        aux
        vmax(aux) = fxy(rx,ry);
        fxy(rx,ry)
        rx
        ry
        aux = aux + 1;
        %criar novo ponto aleatorio
        rx=(rand-0.5)*2*3;
        ry=(rand-0.5)*2*3;

    end

end



hold off

%grafico de x
figure
plot(vx,'r')
%grafico de y
figure
plot (vy,'b')
%grafico dos maximos
figure
plot(vxy,'g')
legend("fxy");
```

Simulated Annealing com 2 funções

```
%Funções
f1 = @(x,y) 3.*(1-x.^2).*exp(-x.^2 -(1+y).^2);
f2 = @(x,y) -10*(x/5 - x.^3 - y.^5).*exp(-x.^2 - y.^2);
f3 = @(x,y) -1/3 * exp(-(x+1).^2 -y.^2);
%função principal
fxy = @(x,y) abs(f1(x,y) + f2(x,y) + f3(x,y));

 %Funções
        g1 = @(x,y) 9.*(1-x.^2).*exp(-x.^2 -(1+y).^2);
        g2 = @(x,y) -5*(x/5 - x.^3 - y.^5).*exp(-x.^2 - y.^2);
        g3 = @(x,y) -1/3 * exp(-(x+1).^2 -y.^2);
        %função principal
        gxy = @(x,y) abs(g1(x,y) + g2(x,y) + g3(x,y));



%%fxy = @(x,y) 3.*(1-x.^2).*exp(-x.^2 -(1+y).^2);

close all;

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


%Geração de ponto aleatorio
rx=(rand-0.5)*2*3;
ry=(rand-0.5)*2*3;

hold on

%parametros da figura
x=linspace(-3,4,100);
y=linspace(-3,4,100);

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


FX=fxy(X,Y);

%desenho da figura
contour(X,Y,FX,20);

%variavel para trocar de função quando estabiliza
n=0;


%%vetores
vx = zeros(20,1);
vy = zeros(20,1);
vxy = zeros(20,1);
vT = zeros(20,1);
vp = zeros(20,1);

%%interações, Temperatura e variavel de vizinhança
```

```
it=1;
t_it = 1;
T=90;
d=(1.5/90)*T;%vizinhaça

while(n ~= 1) %criterio: n iterações
    if (t_it == 10)
        t_it = 0;
        T = T * 0.86;
        if (T < 0)
            T = 0;
        end
    end

    %geração de novos pontos
    new_x= rx + (rand-0.5)*2*d;
    new_y= ry + (rand-0.5)*2*d;

    %limitador de coordenadas
    while new_y < -3
        new_y= ry + (rand-0.5)*2*d;
    end
    while new_x < -3
        new_x= rx + (rand-0.5)*2*d;
    end
    while new_y > 3
        new_y= ry + (rand-0.5)*2*d;
    end
    while new_x > 3
        new_x= rx + (rand-0.5)*2*d;
    end

    %diferença dos pontos
    %calculo da probabilidade apartir dele e da
    %Temperatura da interação
    deltaE= fxy(new_x,new_y) - fxy(rx,ry);
    p = exp(deltaE / T);

     if T < 2.5 %diminui a vizinhança
        d= 0.05;
    end


    if (p < 0)
        p = 0; % probabilidade
    end
    if   deltaE > 0 %maximização
        rx = new_x;
        ry = new_y;
        p=1;
    elseif rand < p
        rx = new_x;
        ry = new_y;
    end

    %Guardar pontos nos vetores
    vx(it) = rx;
    vy(it) = ry;
    vxy(it) = fxy(rx,ry);
```

```
    vT(it) = T;
    vp(it) = p;


    %colocar os pontos no grafico
    plot(rx,ry,'bo')
    hold on


    %estabilização
     if(it>10 && vxy(it-10)==vxy(it))
         maximo_func1 = fxy(rx,ry)
        n = 1;

     end

     t_it = t_it + 1; %iterador temperatura
    it = it + 1; %iterador principal
end

figure

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

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

GX=gxy(X,Y);

contour(X,Y,GX,20)
hold on

t_it = 1;
T=90;
d=(1.5/90)*T;

while(n ~= 2) %criterio: n iterações
    if (t_it == 10)
        t_it = 0;
        T = T * 0.86;
        if (T < 0)
            T = 0;
        end
    end

    %geração de novos pontos
    new_x= rx + (rand-0.5)*2*d;
    new_y= ry + (rand-0.5)*2*d;

    %limitador de coordenadas
    while new_y < -3
        new_y= ry + (rand-0.5)*2*d;
    end
    while new_x < -3
        new_x= rx + (rand-0.5)*2*d;
    end
    while new_y > 3
        new_y= ry + (rand-0.5)*2*d;
    end
```

```
    while new_x > 3
        new_x= rx + (rand-0.5)*2*d;
    end


    %diferença dos pontos
    %calculo da probabilidade apartir dele e da
    %Temperatura da interação
    deltaE= gxy(new_x,new_y) - gxy(rx,ry);
    p = exp(deltaE / T);


   if T < 2.5 %diminui a vizinhança
        d= 0.05;
    end



    if (p < 0)
        p = 0; % probabilidade
    end
    if   deltaE > 0 %maximização
        rx = new_x;
        ry = new_y;
        p=1;
    elseif rand < p
        rx = new_x;
        ry = new_y;
    end

    %Guardar pontos nos vetores
    vx(it) = rx;
    vy(it) = ry;
    vxy(it) = gxy(rx,ry);
    vT(it) = T;
    vp(it) = p;


    %colocar os pontos no grafico
    plot(rx,ry,'go')
    hold on


    %estabilização
     if(it>10 && vxy(it-10)==vxy(it))
         maximo_func2 = gxy(rx,ry)
        n = 2;

     end

     t_it = t_it + 1; %iterador temperatura
    it = it + 1; %iterador principal
end



%graficos de x e y
```

```
figure
plot(vx,'r')
legend("x")

figure
plot (vy,'b')
legend("y")


%grafico dos máximos
figure
plot(vxy,'C')
legend("fxy");

%Grafico da probabilidade
figure
plot(vp,'mo')
legend("p");

%grafico da Temperatura
figure
plot(vT,'yp')
legend("Temperatura");

hold off
```