Inteligência Artificial – Trabalho 2

Algoritmos de Pesquisa

Vitor Alberto de Sá Ribeiro – 78138

Pedro de Carvalho Martins - 76934

Índice

[*Índice de Figuras* 4](#_Toc184494061)

[Resumo 5](#_Toc184494062)

[Introdução 6](#_Toc184494063)

[Subida da Colina 7](#_Toc184494064)

[Conceito 7](#_Toc184494065)

[Implementações 7](#_Toc184494066)

[Características do Código 8](#_Toc184494067)

[Visualização: 8](#_Toc184494068)

[Diferença Principal: 8](#_Toc184494069)

[Exemplos 8](#_Toc184494070)

[Conclusão 9](#_Toc184494071)

[Simulated Annealing 9](#_Toc184494072)

[Conceito 9](#_Toc184494073)

[Implementação 9](#_Toc184494074)

[Definição de Parâmetros e Inicialização: 9](#_Toc184494075)

[Exploração do Espaço de Busca: 10](#_Toc184494076)

[Critério de Aceitação: 10](#_Toc184494077)

[Decaimento da Temperatura: 10](#_Toc184494078)

[Armazenamento e Análise dos Dados: 10](#_Toc184494079)

[Identificação do Melhor Ponto: 11](#_Toc184494080)

[Visualização dos Resultados: 11](#_Toc184494081)

[Resumo do Processo 11](#_Toc184494082)

[Exemplos 11](#_Toc184494083)

[Conclusão 14](#_Toc184494084)

[Caixeiro Viajante 14](#_Toc184494085)

[Conceito 14](#_Toc184494086)

[Implementação 15](#_Toc184494087)

[Carregamento das Cidades 15](#_Toc184494088)

[Configuração do Algoritmo de Simulated Annealing 15](#_Toc184494089)

[Execução do Simulated Annealing 15](#_Toc184494090)

[Resfriamento e Convergência 15](#_Toc184494091)

[Exibição 15](#_Toc184494092)

[Exemplos 15](#_Toc184494093)

[Conclusão 19](#_Toc184494094)

[Conclusão 19](#_Toc184494095)

[Codigo em Anexo 20](#_Toc184494096)

[Subida de Colina Original 20](#_Toc184494097)

[Subida e Colina Reinicialização Múltipla 24](#_Toc184494098)

[Simulated Annealing 27](#_Toc184494099)

[Caixeiro Viajante 32](#_Toc184494100)

[Swapcities\_24 32](#_Toc184494101)

[Pt\_nt\_sul\_30 33](#_Toc184494102)

[Pt\_nt\_sul\_20 35](#_Toc184494103)

[Pt\_nt 37](#_Toc184494104)

[Plotcities\_2024 39](#_Toc184494105)

[Main\_2024 41](#_Toc184494106)

[Geo\_distance 45](#_Toc184494107)

[Distance\_24 46](#_Toc184494108)

# Índice de Figuras

[Figura 1 - Subida Da Colina Original 8](#_Toc184494138)

[Figura 2 - Subida Da Colina Reinicialização Múltipla 9](#_Toc184494139)

[Figura 3 - SA Definições Teste 1 11](#_Toc184494140)

[Figura 4 - SA Teste 1 Plot 1 12](#_Toc184494141)

[Figura 5 - SA Teste 1 Plot 2 12](#_Toc184494142)

[Figura 6 - SA Definições Teste 2 13](#_Toc184494143)

[Figura 7 - SA Teste 2 Plot 1 13](#_Toc184494144)

[Figura 8 - SA Teste 2 Plot 2 14](#_Toc184494145)

[Figura 9 - TSP Definições 16](#_Toc184494146)

[Figura 10 - TSP 30 Cidades Plot 1 16](#_Toc184494147)

[Figura 11- TSP 30 Cidades Plot 2 17](#_Toc184494148)

[Figura 12 - TSP 20 Cidades Plot 1 17](#_Toc184494149)

[Figura 13 - TSP 20 Cidades Plot 2 18](#_Toc184494150)

[Figura 14 - TSP 10 Cidades Plot 1 18](#_Toc184494151)

[Figura 15 - TSP 10 Cidades Plot 2 19](#_Toc184494152)

# Resumo

Este trabalho tem como objetivo a análise e comparação de três abordagens clássicas de resolução de problemas de otimização: Hill Climbing, Simulated Annealing (SA) e a aplicação do Simulated Annealing ao Problema do Caixeiro Viajante (TSP). O TSP é um problema de otimização combinatória clássico, onde se busca a rota mais curta para visitar um conjunto de cidades, retornando à cidade inicial. O trabalho visa entender as vantagens e limitações de cada algoritmo, avaliar a sua eficácia e aplicar o SA ao TSP para verificar seu desempenho em um cenário real.

O Hill Climbing e o Simulated Annealing são métodos de otimização amplamente usados em inteligência artificial e teoria da computação. O primeiro é um algoritmo iterativo que busca encontrar a solução ótima, movendo-se sempre na direção que melhora a solução corrente, mas, em alguns casos, pode ficar preso em mínimos locais. Já o Simulated Annealing é inspirado no processo físico de resfriamento de metais, permitindo que a solução explore novas áreas do espaço de soluções e aceite soluções subótimas com o objetivo de escapar de mínimos locais. A aplicação do SA ao TSP é uma forma de explorar o potencial dessa técnica heurística em um problema clássico de otimização combinatória.

O principal objetivo deste trabalho é comparar os algoritmos Hill Climbing e Simulated Annealing para avaliar o desempenho de ambos em termos de capacidade de encontrar soluções ótimas e a sua robustez frente a problemas de otimização como o TSP. Para isso, o trabalho tem os seguintes objetivos específicos:

Implementar e testar o Hill Climbing e o Simulated Annealing em problemas de otimização.

Aplicar o Simulated Annealing ao TSP e avaliar os resultados obtidos.

Comparar os resultados dos dois algoritmos, destacando suas vantagens, limitações e eficiência em diferentes cenários.

Para atingir os objetivos propostos, foram seguidas as seguintes etapas:

Implementação dos Algoritmos: O Hill Climbing e o Simulated Annealing foram implementados em MATLAB. No caso do TSP, foi criada uma implementação específica do Simulated Annealing, onde o algoritmo foi adaptado para otimizar a rota entre um conjunto de cidades, utilizando um critério de aceitação probabilística baseado na diferença de distância entre soluções sucessivas.

Testes e Comparações: Para avaliar a eficácia dos algoritmos, foram realizadas várias simulações, utilizando um conjunto de 30 cidades representando cidades em Portugal. O desempenho dos algoritmos foi analisado com base na distância total da rota encontrada e na evolução das métricas ao longo das iterações, como a temperatura no caso do SA e a probabilidade de aceitação de novas soluções.

Análise dos Resultados: Os resultados obtidos foram analisados, comparando a capacidade de ambos os algoritmos em encontrar boas soluções e escapar de mínimos locais, bem como a sua eficiência em termos de tempo computacional.

Os resultados demonstraram que o Simulated Annealing superou o Hill Climbing na maioria dos cenários, especialmente em problemas mais complexos como o TSP. O Hill Climbing foi eficaz em encontrar soluções rápidas, mas muitas vezes ficou preso em mínimos locais, resultando em soluções subótimas. Por outro lado, o Simulated Annealing, com sua capacidade de explorar o espaço de soluções de forma mais ampla e aceitar soluções subótimas, conseguiu encontrar soluções mais robustas e mais próximas do ótimo global. No caso do TSP, o SA foi capaz de encontrar rotas eficientes com distâncias otimizadas, demonstrando ser uma técnica mais apropriada para esse tipo de problema.

Em suma, o trabalho confirmou que, para o problema do TSP, o Simulated Annealing é uma abordagem mais eficaz do que o Hill Climbing, oferecendo melhores resultados e maior robustez.

# Introdução

Os métodos de pesquisa estocástica, como **Hill-Climbing** e **Simulated Annealing (SA)**, são usados para resolver problemas de otimização. Este trabalho explora esses algoritmos, aplicando-os na otimização de funções e na resolução do problema do Caixeiro Viajante (TSP).  
O trabalho tem como objetivo implementar e comparar os algoritmos Hill-Climbing e SA na otimização de uma função bidimensional, e aplicar o SA para encontrar a rota mais curta no TSP.  
Utilizaremos Matlab para implementar o Hill-Climbing tradicional e com reinicialização múltipla e o SA na otimização da função alvo. Posteriormente, adaptaremos o SA para o TSP, utilizando as coordenadas de 30, 20 e 10 cidades em Portugal.  
Espera-se que o SA mostre maior eficácia na busca pelo máximo global e que encontre soluções aproximadas eficientes para o TSP, superando as limitações do Hill-Climbing em cenários complexos

# Subida da Colina

Conceito  
O algoritmo de Subida de Colina (Hill Climbing) é uma técnica de otimização iterativa que busca maximizar (ou minimizar) uma função movendo-se na direção dos pontos que oferecem melhor valor da função objetivo. O algoritmo para ao encontrar um máximo local, onde nenhum movimento melhora o valor da função.

## Implementações

Sem Reinicialização Múltipla  
Neste caso, o algoritmo começa de um ponto aleatório e segue avaliando pequenos passos para a direita e para a esquerda dentro do intervalo permitido.

Funcionamento:

* + Um passo é escolhido aleatoriamente entre um valor mínimo (StepMin) e um valor máximo (StepMax).
  + O valor da função objetivo é avaliado no ponto atual e nos pontos deslocados.
  + O algoritmo move-se na direção que aumenta o valor da função.
  + Quando um máximo local é encontrado, o algoritmo para e retorna o melhor ponto identificado.

Limitação: Se o algoritmo cair em um máximo local, ele não consegue explorar outras regiões do espaço de busca.

Com Reinicialização Múltipla  
Aqui, o algoritmo segue a mesma lógica básica, mas, ao encontrar um máximo local, ele reinicializa para um ponto aleatório dentro do intervalo definido e continua a busca.

Funcionamento:

* + Após identificar um máximo local, a variável xProbe é redefinida para um novo ponto aleatório.
  + O algoritmo repete o processo até alcançar o número máximo de iterações (It).

Vantagem: A reinicialização permite explorar múltiplos máximos locais, aumentando as chances de encontrar o máximo global.

## Características do Código

Ambos os algoritmos utilizam a mesma função alvo:

f(x)=4⋅(sin(5πx+0.5))^6 ⋅exp(log2((x−0.8)^2))

definida no intervalo [0,1.6].

### Visualização:

* + O código inclui gráficos que mostram a evolução dos valores de x e f(x) ao longo das iterações, assim como os pontos percorridos no gráfico da função.
  + Destaca-se o maior ponto encontrado com um marcador especial.

### Diferença Principal:

* + O algoritmo **sem reinicialização** para na primeira subida a um máximo local.
  + O algoritmo **com reinicialização** continua explorando até atingir o limite de iterações.

## Exemplos

Após 5 tentativas, o melhor resultado com a subida de colina original foi o seguinte:

Uma imagem com texto, captura de ecrã, diagrama, Gráfico

Descrição gerada automaticamente

Figura 1 - Subida Da Colina Original

No entanto, apenas uma tentativa com o Subida de colina com reinicialização múltipla, resultou na melhor solução.

Uma imagem com texto, captura de ecrã, diagrama, Gráfico

Descrição gerada automaticamente

Figura 2 - Subida Da Colina Reinicialização Múltipla

Conclusão  
O método com reinicialização é mais robusto, pois melhora a exploração do espaço de busca, conseguindo assim resolver o problema dos máximos local da implementação original.

# Simulated Annealing

## Conceito

O Simulated Annealing (SA) é um algoritmo de otimização inspirado no processo de recozimento térmico em metalurgia. Ele tenta encontrar o máximo (ou mínimo) global de uma função, explorando o espaço de busca de maneira eficiente. Ele evita ficar preso em máximos/mínimos locais ao permitir movimentos "maus" (soluções piores) de forma controlada, com base numa "temperatura" que decresce gradualmente.

## Implementação

Ambos os algoritmos utilizam a mesma função alvo:

f(x)=4⋅(sin(5πx+0.5)) ^6 ⋅exp(log2((x−0.8)^2))

definida no intervalo [0,1.6].

### Definição de Parâmetros e Inicialização:

* + Define o **número de iterações externas** (It) e internas (ItInternas) para controlar o número de avaliações.
  + Define o **passo máximo** (StepMax) que determina o tamanho do deslocamento para o próximo ponto.
  + Estabelece uma **temperatura inicial** (Temp) e uma taxa de decaimento (TempDecay) para reduzir gradualmente a probabilidade de aceitar soluções piores.
  + Um ponto inicial é escolhido aleatoriamente dentro do intervalo permitido da função objetivo ([0,1.6])

### Exploração do Espaço de Busca:

* + Para cada iteração:
    - Avalia o valor da função objetivo (f(x)) no ponto atual.
    - Gera um novo ponto com um deslocamento aleatório dentro do limite definido por StepMax.
    - Garante que o novo ponto está dentro do intervalo permitido.

### Critério de Aceitação:

* + Calcula a diferença de energia (ΔE) entre o valor da função no ponto atual e no novo ponto.
  + **Aceita o novo ponto** em dois casos:
    - Se o novo ponto é melhor (ΔE>0).
    - Se o novo ponto é pior, mas passa no teste probabilístico (e^(ΔE/Temp)>random(0,1)).
  + Este segundo critério permite que o algoritmo escape de máximos locais.

### Decaimento da Temperatura:

* + Após cada iteração externa, a temperatura é reduzida de acordo com o fator TempDecay. Isto diminui gradualmente a probabilidade de aceitar soluções piores, focando mais na exploração inicial e no refinamento final.

### Armazenamento e Análise dos Dados:

* + São armazenados os valores de:
    - **Pontos visitados** (x e f(x)).
    - **Diferença de energia** (ΔE) entre os pontos.
    - **Probabilidade de aceitação** (e^(ΔE/Temp)).
  + Isto permite a análise da evolução do processo de otimização.

### Identificação do Melhor Ponto:

* + Durante todo o processo, o maior valor de f(x) encontrado é armazenado, junto com o valor correspondente de x.

### Visualização dos Resultados:

* + Gráficos são criados para:
    - Mostrar os **pontos percorridos** em relação à função.
    - Exibir a evolução dos valores de x e f(x) ao longo das iterações.
    - Representar a **probabilidade de aceitação** e as diferenças de energia.

### Resumo do Processo

* O algoritmo começa explorando amplamente a função devido à alta temperatura.
* Conforme a temperatura diminui, a exploração torna-se mais refinada, focando na área em torno dos melhores valores encontrados.
* A combinação de exploração inicial ampla e exploração refinada final permite encontrar o máximo global da função, mesmo em presença de máximos locais.

## Exemplos

Neste algoritmo temos alguns parâmetros que podemos alterar para mudar o seu funcionamento.

**Começamos com as seguintes definições:**

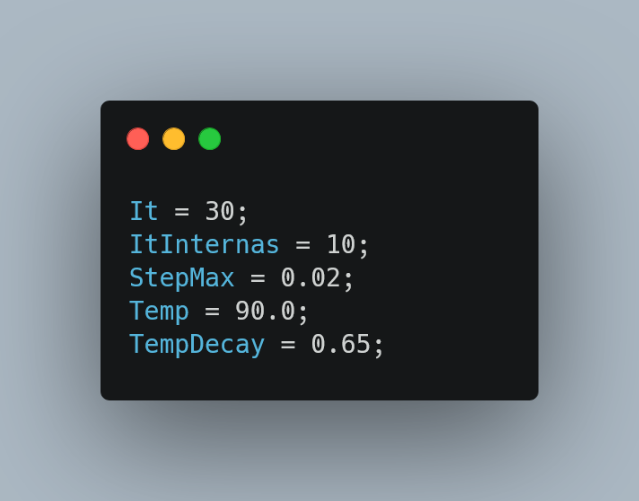


Figura 3 - SA Definições Teste 1

Obtivemos então o seguinte resultado:

Uma imagem com texto, captura de ecrã, file, diagrama

Descrição gerada automaticamente

Figura 4 - SA Teste 1 Plot 1

Uma imagem com texto, file, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 5 - SA Teste 1 Plot 2

Alterando as definições para:

Uma imagem com texto, captura de ecrã, Tipo de letra, eletrónica

Descrição gerada automaticamente

Figura 6 - SA Definições Teste 2

Resultado Obtido:

Uma imagem com texto, captura de ecrã, file, Gráfico

Descrição gerada automaticamente

Figura 7 - SA Teste 2 Plot 1

Uma imagem com texto, captura de ecrã, Gráfico, diagrama

Descrição gerada automaticamente

Figura 8 - SA Teste 2 Plot 2

## Conclusão

Podemos concluir que o Simulated Annealing (SA) oferece uma grande flexibilidade e personalização, permitindo que o algoritmo se adapte a diferentes problemas e cenários por meio da modulação de suas variáveis de entrada, como a temperatura inicial, a taxa de decaimento, o passo máximo e o número de iterações. Isso torna o SA uma abordagem eficiente e robusta para otimizar funções complexas, especialmente quando se trata de problemas com múltiplos ótimos locais ou funções não lineares.

# Caixeiro Viajante

## Conceito

O problema do Caixeiro Viajante (ou TSP - Travelling Salesman Problem) é um clássico problema de otimização combinatória onde o objetivo é encontrar o caminho mais curto que passa por um conjunto de cidades, visitando cada cidade uma única vez e retornando à cidade de origem. Este problema é conhecido por sua complexidade computacional, sendo NP-difícil, ou seja, não existe um algoritmo eficiente para resolvê-lo em tempo polinomial.

Neste programa, utilizamos o algoritmo de Simulated Annealing (SA) para resolver o problema do Caixeiro Viajante. O SA é uma técnica de otimização inspirada no processo de resfriamento de metais e é particularmente útil para encontrar ótimos globais em problemas com muitos ótimos locais. Ao longo da execução, o algoritmo começa com uma "temperatura" alta, permitindo grandes mudanças na solução, e gradualmente diminui a temperatura, restringindo as mudanças até que uma solução de baixo custo seja encontrada.

## Implementação

Carregamento das Cidades

O programa começa com a carga de 30 20 ou 10 cidades localizadas em Portugal. Essas cidades são representadas por coordenadas geográficas, e o objetivo é encontrar o caminho que as conecta com a menor distância total.

Configuração do Algoritmo de Simulated Annealing

A temperatura inicial (T), a temperatura mínima (T\_min), a taxa de resfriamento (alpha) e o número máximo de iterações por temperatura (max\_iter) são definidos. Esses parâmetros controlam o comportamento do algoritmo e determinam sua eficiência e capacidade de exploração do espaço de soluções.

### Execução do Simulated Annealing

O algoritmo começa com uma solução inicial e tenta melhorar a rota ao trocar aleatoriamente duas cidades em cada iteração. A aceitação de uma nova solução é determinada pela diferença de custo (distância) entre a solução atual e a nova, com uma probabilidade que depende da temperatura. Caso a nova solução seja melhor, ela é aceite; caso contrário, ela pode ser aceita com base numa probabilidade que diminui à medida que a temperatura diminui.

### Resfriamento e Convergência

A cada iteração, a temperatura é reduzida, permitindo que o algoritmo se concentre mais em otimizar a solução e escapar de soluções subótimas. Esse processo continua até que a temperatura alcance o valor mínimo.

### Exibição

Após a execução do algoritmo, o programa exibe a melhor rota encontrada e a sua distância total. Também gera gráficos que mostram a evolução da temperatura, da distância da melhor solução, da probabilidade de aceitação e da variação de distância (delta E) ao longo do processo.

### Exemplos

Nestes testes será utilizada a seguinte configuração inicial:

Uma imagem com texto, captura de ecrã, eletrónica, Dispositivo eletrónico

Descrição gerada automaticamente

Figura 9 - TSP Definições

É importante entender que as soluções são sempre diferentes então iremos mostrar o primeiro resultado de cada teste.

Primeiro Resultado 30 Cidades:

Uma imagem com file, diagrama, Gráfico, captura de ecrã

Descrição gerada automaticamente

Figura 10 - TSP 30 Cidades Plot 1

Uma imagem com texto, captura de ecrã, diagrama, Gráfico

Descrição gerada automaticamente

Figura 11- TSP 30 Cidades Plot 2

Primeiro Resultado 20 Cidades:

Uma imagem com diagrama, file, Gráfico, ladeira

Descrição gerada automaticamente

Figura 12 - TSP 20 Cidades Plot 1

Uma imagem com texto, captura de ecrã, diagrama, Gráfico

Descrição gerada automaticamente

Figura 13 - TSP 20 Cidades Plot 2

Primeiro Resultado 10 Cidades:

Uma imagem com file, diagrama, Gráfico, texto

Descrição gerada automaticamente

Figura 14 - TSP 10 Cidades Plot 1

Uma imagem com texto, captura de ecrã, Gráfico, diagrama

Descrição gerada automaticamente

Figura 15 - TSP 10 Cidades Plot 2

## Conclusão

O algoritmo de Simulated Annealing aplicado ao TSP, neste programa, demonstrou ser uma técnica robusta e adaptável, capaz de explorar o espaço de soluções e encontrar boas soluções em tempo razoável. A capacidade do SA de aceitar soluções subótimas com base em uma probabilidade controlada pela temperatura permite que ele escape de mínimos locais, o que é essencial para evitar que o algoritmo fique preso em soluções subótimas.

Os resultados obtidos mostram que, apesar de não garantir uma solução ótima global, o SA proporciona uma solução de alta qualidade, especialmente para problemas de maior escala. A flexibilidade do algoritmo em termos de parâmetros como temperatura inicial, taxa de resfriamento e número de iterações permite que ele seja ajustado de acordo com a complexidade do problema, oferecendo uma boa relação entre tempo de execução e qualidade da solução.

Em resumo, o Simulated Annealing é uma abordagem poderosa para o TSP, proporcionando soluções eficientes para instâncias de diferentes tamanhos e complexidades. Embora não ofereça garantias de otimização global, ele é uma excelente escolha para problemas práticos em que a rapidez de execução e a qualidade da solução são fatores determinantes.

# Conclusão

Neste trabalho, foram analisadas três abordagens clássicas para a resolução de problemas de otimização: Hill Climbing, Simulated Annealing (SA) e a aplicação do Simulated Annealing ao problema do Caixeiro Viajante (TSP). Através de simulações e análises, foi possível avaliar as vantagens, limitações e eficácia de cada método, proporcionando uma visão mais clara sobre as suas aplicações em diferentes cenários.

O algoritmo Hill Climbing, embora simples e intuitivo, demonstrou ser limitado pela sua tendência a ficar preso em mínimos locais. Como se trata de uma técnica greedy, o Hill Climbing prioriza a melhoria imediata, mas a falta de capacidade para explorar outras áreas do espaço de soluções dificulta a obtenção de resultados globais ótimos em problemas mais complexos.

Já o Simulated Annealing (SA) revelou-se significativamente mais robusto e eficaz, graças à sua capacidade de aceitar soluções subótimas durante a execução, permitindo escapar de mínimos locais e realizar uma busca mais ampla pelo espaço de soluções. Esta característica faz do SA uma técnica especialmente adequada para problemas onde o espaço de soluções é vasto e não linear, como o TSP. No caso do Caixeiro Viajante, o SA foi capaz de encontrar rotas eficientes com distâncias otimizadas em tempo razoável, mesmo sem garantir uma solução global ótima, devido à natureza heurística do algoritmo.

A comparação entre Hill Climbing e Simulated Annealing evidenciou que o SA apresenta um desempenho superior em problemas mais complexos. A sua flexibilidade na exploração de soluções, controlada por parâmetros como a temperatura inicial, a taxa de resfriamento e o número de iterações, permite uma adaptação mais eficiente a diferentes problemas.

Em suma, este trabalho demonstrou que, para problemas como o TSP, o Simulated Annealing é uma abordagem mais eficaz do que o Hill Climbing, oferecendo resultados mais robustos e próximos do ótimo global. Além disso, a aplicação de técnicas heurísticas como o SA a problemas clássicos realça a sua utilidade na resolução de desafios de otimização. Para trabalhos futuros, sugere-se a investigação de configurações mais avançadas de parâmetros ou a integração com outras técnicas, como algoritmos genéticos, para melhorar ainda mais o desempenho em problemas de maior escala e complexidade.

# Codigo em Anexo

## Subida de Colina Original

close all;

clear all;

% Variáveis do programa

It = 300; % Número máximo de iterações

StepMax = 0.02; % Passo máximo inicial

StepMin = 0.001; % Passo mínimo para evitar passos muito pequenos

% Inicialização da matriz para guardar todos os pontos percorridos

todos\_pontos = zeros(2, It);

contador = 1;

% Variáveis para armazenar o maior ponto

maiorX = 0;

maiorY = -Inf;

% Função e dados para o gráfico

f = @(x) 4 \* (sin(5 \* pi \* x + 0.5)).^6 .\* exp(log2((x - 0.8).^2));

x = linspace(0, 1.6, 200);

y = f(x);

% Plot da função

figure;

subplot(1, 3, 1); % 1 row, 3 columns, position 2

plot(x, y, 'b');

hold on;

title('Subida de Colina (Hill Climbing) - Máximos Locais Encontrados');

xlabel('x');

ylabel('f(x)');

% Ponto inicial aleatório

xProbe = rand \* 1.6;

for i = 1:It

% Passo aleatório para direita e esquerda com passo adaptativo

xStep = max(StepMin, rand \* StepMax);

xProbeR = xProbe + xStep;

xProbeL = xProbe - xStep;

% Garantir que xProbeR e xProbeL fiquem dentro do intervalo [0, 1.6]

xProbeR = max(0, min(1.6, xProbeR));

xProbeL = max(0, min(1.6, xProbeL));

% Avaliar os valores de f(x) no ponto atual e nas direções direita e esquerda

currentValue = f(xProbe);

valueR = f(xProbeR);

valueL = f(xProbeL);

% Guardar o ponto atual na matriz de todos os pontos percorridos

todos\_pontos(1, contador) = xProbe;

todos\_pontos(2, contador) = currentValue;

contador = contador + 1;

% Determinar a direção da subida

if (valueR > currentValue)

xProbe = xProbeR;

elseif (valueL > currentValue)

xProbe = xProbeL;

elseif (valueL == valueR)

xProbe = xProbeR; % Em caso de empate, move-se para a direita

else

% Se nenhum passo aumenta f(x), um máximo local é alcançado

if currentValue > maiorY

maiorX = xProbe;

maiorY = currentValue;

end

end

end

% Exibir a matriz de todos os pontos percorridos

disp('Topo da Colina:');

maiorX

maiorY

% Plot de todos os pontos percorridos

plot(todos\_pontos(1, 1:contador-1), todos\_pontos(2, 1:contador-1), 'go');

% Destacar o maior ponto com uma cor diferente

plot(maiorX, maiorY, 'ro', 'MarkerSize', 8, 'LineWidth', 2);

hold off;

% Plot dos valores de Y (f(x)) ao longo do tempo

subplot(1, 3, 2); % 1 row, 3 columns, position 2

plot(1:contador-1, todos\_pontos(2, 1:contador-1), 'b.-');

title('Evolução dos valores de Y ao longo do tempo');

xlabel('Iteração');

ylabel('f(x)');

% Plot dos valores de X ao longo do tempo

subplot(1, 3, 3); % 1 row, 3 columns, position 2

plot(1:contador-1, todos\_pontos(1, 1:contador-1), 'r.-');

title('Evolução dos valores de X ao longo do tempo');

xlabel('Iteração');

ylabel('x');

## Subida e Colina Reinicialização Múltipla

close all;

clear all;

% Variáveis do programa

It = 300; % Número máximo de iterações

StepMax = 0.02; % Passo máximo inicial

StepMin = 0.001; % Passo mínimo para evitar passos muito pequenos

% Inicialização da matriz para guardar todos os pontos percorridos

todos\_pontos = zeros(2, It);

contador = 1;

% Variáveis para armazenar o maior ponto

maiorX = 0;

maiorY = -Inf;

% Função e dados para o gráfico

f = @(x) 4 \* (sin(5 \* pi \* x + 0.5)).^6 .\* exp(log2((x - 0.8).^2));

x = linspace(0, 1.6, 200);

y = f(x);

% Criar figura com subplots para exibir todos os gráficos lado a lado

figure;

% Subplot 1: Plot da função e pontos percorridos

subplot(1, 3, 1); % 1 row, 3 columns, position 1

plot(x, y, 'b');

hold on;

title('Função e Pontos Percorridos');

xlabel('x');

ylabel('f(x)');

% Ponto inicial aleatório

xProbe = rand \* 1.6;

for i = 1:It

% Passo aleatório para direita e esquerda com passo adaptativo

xStep = max(StepMin, rand \* StepMax);

xProbeR = xProbe + xStep;

xProbeL = xProbe - xStep;

% Garantir que xProbeR e xProbeL fiquem dentro do intervalo [0, 1.6]

xProbeR = max(0, min(1.6, xProbeR));

xProbeL = max(0, min(1.6, xProbeL));

% Avaliar os valores de f(x) no ponto atual e nas direções direita e esquerda

currentValue = f(xProbe);

valueR = f(xProbeR);

valueL = f(xProbeL);

% Guardar o ponto atual na matriz de todos os pontos percorridos

todos\_pontos(1, contador) = xProbe;

todos\_pontos(2, contador) = currentValue;

contador = contador + 1;

% Determinar a direção da subida

if (valueR > currentValue)

xProbe = xProbeR;

elseif (valueL > currentValue)

xProbe = xProbeL;

elseif (valueL == valueR)

xProbe = xProbeR; % Em caso de empate, move-se para a direita

else

% Se nenhum passo aumenta f(x), um máximo local é alcançado

if currentValue > maiorY

maiorX = xProbe;

maiorY = currentValue;

end

% Reinicializar o xProbe para outro ponto aleatório

xProbe = rand \* 1.6;

end

end

% Exibir a matriz de todos os pontos percorridos

disp('Todos os pontos percorridos:');

disp(todos\_pontos(:, 1:contador-1));

% Plot dos pontos percorridos

plot(todos\_pontos(1, 1:contador-1), todos\_pontos(2, 1:contador-1), 'go');

% Destacar o maior ponto encontrado

plot(maiorX, maiorY, 'ro', 'MarkerSize', 8, 'LineWidth', 2);

hold off;

% Subplot 2: Evolução dos valores de Y ao longo do tempo

subplot(1, 3, 2); % 1 row, 3 columns, position 2

plot(1:contador-1, todos\_pontos(2, 1:contador-1), 'b.-');

title('Evolução dos valores de Y ao longo do tempo');

xlabel('Iteração');

ylabel('f(x)');

% Subplot 3: Evolução dos valores de X ao longo do tempo

subplot(1, 3, 3); % 1 row, 3 columns, position 3

plot(1:contador-1, todos\_pontos(1, 1:contador-1), 'r.-');

title('Evolução dos valores de X ao longo do tempo');

xlabel('Iteração');

ylabel('x');

## Simulated Annealing

close all;

clear all;

% Variáveis do programa

It = 30; % Número máximo de iterações

ItInternas = 10;

StepMax = 0.02; % Passo máximo

Temp = 90.0; % Temperatura inicial

TempDecay = 0.65; % Taxa de decaimento da temperatura

% Inicialização para guardar todos os pontos percorridos

todos\_pontos = zeros(2, It\*ItInternas);

Probabilidade = zeros(1, It\*ItInternas);

DeltaEnergia = zeros(1, It\*ItInternas);

contador = 1;

% Variáveis para armazenar o maior ponto encontrado

maiorX = 0;

maiorY = -Inf;

% Função e dados para o gráfico

f = @(x) 4 \* (sin(5 \* pi \* x + 0.5)).^6 .\* exp(log2((x - 0.8).^2));

x = linspace(0, 1.6, 200);

y = f(x);

% Criar figura com subplots para exibir todos os gráficos lado a lado

figure;

% Subplot 1: Plot da função e pontos percorridos

subplot(3,1 , 1); % 1 row, 3 columns, position 1

plot(x, y, 'b');

hold on;

title('Simulated Annealing - Pontos Percorridos');

xlabel('x');

ylabel('f(x)');

% Ponto inicial aleatório

xProbe = rand \* 1.6;

for i = 1:It

for j = 1:ItInternas

% Calcular f(x) no ponto atual

currentValue = f(xProbe);

% Armazenar o ponto atual na matriz de todos os pontos percorridos

todos\_pontos(1, contador) = xProbe;

todos\_pontos(2, contador) = currentValue;

contador = contador + 1;

% Atualizar o maior ponto encontrado

if currentValue > maiorY

maiorX = xProbe;

maiorY = currentValue;

end

% Passo aleatório para a direita e esquerda

xStep = (rand \* 2 - 1) \* StepMax;

xNew = xProbe + xStep;

% Garantir que xNew está dentro do intervalo [0, 1.6]

xNew = max(0, min(1.6, xNew));

% Calcular f(x) no novo ponto e a diferença de energia

newValue = f(xNew);

deltaE = newValue - currentValue;

Probabilidade(contador) = exp(-abs(deltaE)/ Temp ); %Adicionar a vetor

DeltaEnergia(contador) = deltaE; %% adicionar a Vetor

% Aceitar novo ponto com base na temperatura

if deltaE > 0 || (exp(deltaE / Temp) > rand)

xProbe = xNew; % Mover para o novo ponto

end

end

% Reduzir a temperatura

Temp = Temp \* TempDecay;

end

% Exibir a matriz de todos os pontos percorridos

disp('Todos os pontos percorridos:');

disp(todos\_pontos(:, 1:contador-1));

% Plot dos pontos percorridos

plot(todos\_pontos(1, 1:contador-1), todos\_pontos(2, 1:contador-1), 'go');

% Destacar o maior ponto encontrado

plot(maiorX, maiorY, 'ro', 'MarkerSize', 8, 'LineWidth', 2);

hold off;

% Subplot 2: Evolução dos valores de Y ao longo do tempo

subplot(3, 1, 2); % 1 row, 3 columns, position 2

plot(1:contador-1, todos\_pontos(2, 1:contador-1), 'b.-');

title('Evolução dos valores de Y ao longo do tempo');

xlabel('Iteração');

ylabel('f(x)');

% Subplot 3: Evolução dos valores de X ao longo do tempo

subplot(3, 1, 3); % 1 row, 3 columns, position 3

plot(1:contador-1, todos\_pontos(1, 1:contador-1), 'r.-');

title('Evolução dos valores de X ao longo do tempo');

xlabel('Iteração');

ylabel('x');

figure;

% Subplot 2: Evolução dos valores de Probabilidade ao longo de tempo

subplot(2, 1, 1); % 1 row, 3 columns, position 2

plot(1:contador-1, Probabilidade(1:contador-1), 'b.-');

title('Evolução dos valores de Probabilidade ao longo do tempo');

xlabel('Iteração');

ylabel('Probabilidade');

% Subplot 3: Evolução dos valores de DeltaE ao longo do tempo

subplot(2, 1, 2); % 1 row, 3 columns, position 3

plot(1:contador-1, DeltaEnergia(1:contador-1), 'r.-');

title('Evolução dos valores de DeltaE ao longo do tempo');

xlabel('DeltaE');

ylabel('x');

## Caixeiro Viajante

### Swapcities\_24

% swapcities\_24

% returns a set of m cities where n cities are randomly swaped.

function s = swapcities\_24(inputcities,n)

% Stores inputcities

s = inputcities;

for i = 1 : n

% city\_1- random number betweeen 0-nº of cities

city\_1 = round(length(inputcities)\*rand(1));

if city\_1 < 1

city\_1 = 1;

end

% city\_2- random number betweeen 0-nº of cities

city\_2 = round(length(inputcities)\*rand(1));

if city\_2 < 1

city\_2 = 1;

end

% temp variable for swapping city\_1 with city\_2 coordinates

temp = s(:,city\_1);

s(:,city\_1) = s(:,city\_2);

s(:,city\_2) = temp;

end

### Pt\_nt\_sul\_30

% PMO

% UTAD 2017

clear all;

% 1 Bragança 41N49 6W45

% 2 Vila Real 41N18 7W45

% 3 Chaves 41N44 7W28

% 4 Viana do Castelo 41N42 8W50

% 5 Braga 41N33 8W26

% 6 Aveiro 40N38 8W39

% 7 Porto 41N11 8W36

% 8 Viseu 40N39 7W55

% 9 Lamego 41N06 7W49

% 10 Guimarães 41N27 8W18

% 11 Coimbra 40N12 8W25

% 12 Faro 37N01 7W56

% 13 Évora 38N34 7W54

% 14 Lisboa 38N43 9W10

% 15 Portalegre 39N17 7W26

% 16 Tavira 37N07 7W39

% 17 Sagres 37N00 8W56

% 18 Setúbal 38N32 8W54

% 19 Guarda 40N32 7W16

% 20 Santarém 39N14 8W41

% 21 Beja 38N01 7W52

% 22 Sines 37N57 8W52

% 23 Covilhã 40N17 7W30

% 24 Tomar 39N36 8W25

% 25 Águeda 40N34 8W27

% 26 Leiria 39N45 8W48

% 27 Castelo Branco 39N49 7W30

% 28 Elvas 38N53 7W10

% 29 Miranda do Douro 41N30 6W16

% 30 Sintra 38N48 9W23

temp\_x = [ 1 41.49 6.45

2 41.18 7.45

3 41.44 7.28

4 41.42 8.50

5 41.33 8.26

6 40.38 8.39

7 41.11 8.36

8 40.39 7.55

9 41.06 7.49

10 41.27 8.18

11 40.12 8.25

12 37.01 7.56

13 38.34 7.54

14 38.43 9.10

15 39.17 7.26

16 37.07 7.39

17 37.00 8.56

18 38.32 8.54

19 40.32 7.16

20 39.14 8.41

21 38.01 7.52

22 37.57 8.52

23 40.17 7.30

24 39.36 8.25

25 40.34 8.27

26 39.45 8.48

27 39.49 7.30

28 38.53 7.10

29 41.30 6.16

30 38.48 9.23

];

cities = [temp\_x(:,2)';temp\_x(:,3)'];

### Pt\_nt\_sul\_20

% PMO

% 20 Portuguese Cities Coordinates

clear all;

% 1 Bragança 41N49 6W45

% 2 Vila Real 41N18 7W45

% 3 Chaves 41N44 7W28

% 4 Viana do Castelo 41N42 8W50

% 5 Braga 41N33 8W26

% 6 Aveiro 40N38 8W39

% 7 Porto 41N11 8W36

% 8 Viseu 40N39 7W55

% 9 Lamego 41N06 7W49

% 10 Guimarães 41N27 8W18

% 11 Coimbra 40N12 8W25

% 12 Faro 37N01 7W56

% 13 Évora 38N34 7W54

% 14 Lisboa 38N43 9W10

% 15 Portalegre 39N17 7W26

% 16 Tavira 37N07 7W39

% 17 Sagres 37N00 8W56

% 18 Setúbal 38N32 8W54

% 19 Guarda 40N32 7W16

% 20 Santarém 39N14 8W41

temp\_x = [ 1 41.49 6.45

2 41.18 7.45

3 41.44 7.28

4 41.42 8.50

5 41.33 8.26

6 40.38 8.39

7 41.11 8.36

8 40.39 7.55

9 41.06 7.49

10 41.27 8.18

11 40.12 8.25

12 37.01 7.56

13 38.34 7.54

14 38.43 9.10

15 39.17 7.26

16 37.07 7.39

17 37.00 8.56

18 38.32 8.54

19 40.32 7.16

20 39.14 8.41

];

cities = [temp\_x(:,2)';temp\_x(:,3)'];

### Pt\_nt

% Set of 14 cities

% Mostly in the North of Portugal

% Paulo Moura Oliveira, 2017

%-----------------------------------------------------------------------

% 1 Bragança 41N49 6W45

% 2 Vila Real 41N18 7W45

% 3 Chaves 41N44 7W28

% 4 Viana do Castelo 41N42 8W50

% 5 Braga 41N33 8W26

% 6 Aveiro 40N38 8W39

% 7 Porto 41N11 8W36

% 8 Viseu 40N39 7W55

% 9 Lamego 41N06 7W49

% 10 Águeda 40N34 8W27

% 11 Régua 41N10 7W47

% 12 Guimarães 41N27 8W18

% 13 Valença 42N02 8W38

% 14 Barcelos 41N32 8W37

%-----------------------------------------------------------------------

% Lati and Longit\\

temp\_x = [ 1 41.49 6.45

2 41.18 7.45

3 41.44 7.28

4 41.42 8.50

5 41.33 8.26

6 40.38 8.39

7 41.11 8.36

8 40.39 7.55

9 41.06 7.49

10 40.34 8.27

11 41.10 7.47

12 41.27 8.18

13 42.02 8.38

14 41.32 8.37];

% Stores the cities coordinates in a matrix

cities = [temp\_x(:,2)';temp\_x(:,3)'];

### Plotcities\_2024

% This function plots the cities with coordinates in

% the array inputcities(2xn)

% rows - coordinates in the form of latitute and longitude

% columns - cities

% set\_id - ste identification to adjust plotting scale

function f = plotcities\_2024(inputcities,set\_id)

shg % Show graph window

% temp\_1 = plot(inputcities(1,:),inputcities(2,:),'b\*');

% set(temp\_1,'erasemode','none');

clf

temp\_2 = line(inputcities(1,:),inputcities(2,:),'Marker','o','Markersize',6,'Markerfacecolor','r');

set(temp\_2,'color','r');

xlabel('Lati.')

ylabel('Longi.')

% -------------------------------------------------------------------

% In the case of cities in the North Portugal you can use the following

if set\_id == 1

axis([40.2 42.3 6.2 8.7])

elseif set\_id == 2

%--------------------------------------------------------------------

% In the case of cities all over Portugal you can use the following

axis([36.5 42 6 9.5])

else

fprintf(1,'Error in the set\_id number');

end

% Original----------------------------------------

x = [inputcities(1,1) inputcities(1,length(inputcities))];

y = [inputcities(2,1) inputcities(2,length(inputcities))];

temp\_3 = line(x,y);

set(temp\_3,'color','r');

### Main\_2024

clc % Clear screen

clear all; % Clear all variables from workspace

close all; % Close all figures

%------------------------------------------------------------------------

% Loading 30 cities in Portugal

%pt\_nt\_sul\_30;

%pt\_nt\_sul\_20;

pt\_nt;

set\_id = 2;

% Input Settings

cities = swapcities\_24(cities , size(cities,2));

inputcities = cities;

% Configurações do Simulated Annealing

T = 10000; % Temperatura inicial

T\_min = 0.001; % Temperatura mínima

alpha = 0.99; % Taxa de resfriamento

max\_iter = 10; % Iterações máximas por temperatura

% Inicializar solução inicial

current\_cities = inputcities;

best\_cities = current\_cities;

best\_distance = distance\_24(current\_cities);

% Arrays para armazenar dados das iterações

temperatures = [];

distances = [];

acceptance\_probs = [];

deltaE\_values = [];

% Definicao do Plot

figure;

hold off

plotcities\_2024(best\_cities, set\_id);

title('Melhor rota encontrada com Simulated Annealing');

% Algoritmo de Simulated Annealing

iter = 1; % Contador de iterações

while T > T\_min

for i = 1:max\_iter

% Gera uma nova rota trocando duas cidades

new\_cities = swapcities\_24(current\_cities, 2); % Troca 2 cidades aleatoriamente

% Calcula a diferença de custo (distância)

current\_distance = distance\_24(current\_cities);

new\_distance = distance\_24(new\_cities);

delta = new\_distance - current\_distance;

% Probabilidade de aceitação

prob = exp(-delta / T);

% Armazenar dados da iteração

temperatures(end + 1) = T;

distances(end + 1) = best\_distance;

acceptance\_probs(end + 1) = max(0, min(prob, 1)); % Garantir que esteja no intervalo [0, 1]

deltaE\_values(end + 1) = delta;

% Critério de aceitação

if delta < 0 || rand() < prob

current\_cities = new\_cities;

% Atualiza a melhor solução encontrada

if new\_distance < best\_distance

best\_cities = new\_cities;

best\_distance = new\_distance;

end

end

iter = iter + 1; % Incrementa o contador

end

% Resfriamento

T = T \* alpha;

end

% Resultados finais

fprintf(1, 'A melhor rota encontrada para %d cidades tem uma distância total de %4.2f Km\n', length(inputcities), best\_distance);

% Plot da melhor rota encontrada

plotcities\_2024(best\_cities, set\_id);

% Figura com evolução de métricas

figure;

subplot(2, 2, 1);

plot(temperatures);

title('Temperatura');

xlabel('Iterações');

ylabel('Temperatura');

subplot(2, 2, 2);

plot(distances);

title('Melhor Distância');

xlabel('Iterações');

ylabel('Distância');

subplot(2, 2, 3);

plot(acceptance\_probs);

title('Probabilidade de Aceitação');

xlabel('Iterações');

ylabel('Probabilidade');

subplot(2, 2, 4);

plot(deltaE\_values);

title('DeltaE (Variação de Distância)');

xlabel('Iterações');

ylabel('DeltaE');

### Geo\_distance

% ----------------------------------------------------------

% Function to evaluate the geographical distance

% Conversion from coordinates

% Inputs: [ lat1, long1], [lat2, long2]

%-----------------------------------------------------------

function [dist] = geo\_distance(city\_1,city\_2)

% City 1

x\_graus\_city\_1 = fix(city\_1(1));

min\_x\_city\_1 = city\_1(1)-x\_graus\_city\_1;

lati\_city\_1=pi\*(x\_graus\_city\_1 + 5\*min\_x\_city\_1/3)/180;

y\_graus\_city\_1 = fix(city\_1(2));

min\_y\_city\_1 = city\_1(2)-y\_graus\_city\_1;

longi\_city\_1=pi\*(y\_graus\_city\_1 + 5\*min\_y\_city\_1/3)/180;

% City 2

x\_graus\_city\_2 = fix(city\_2(1));

min\_x\_city\_2 = city\_2(1)-x\_graus\_city\_2;

lati\_city\_2=pi\*(x\_graus\_city\_2 + 5\*min\_x\_city\_2/3)/180;

y\_graus\_city\_2 = fix(city\_2(2));

min\_y\_city\_2 = city\_2(2)-y\_graus\_city\_2;

longi\_city\_2=pi\*(y\_graus\_city\_2 + 5\*min\_y\_city\_2/3)/180;

RRR = 6378.388; % Idealized sphere radius

q1 = cos( longi\_city\_1 - longi\_city\_2 );

q2 = cos( lati\_city\_1 - lati\_city\_2 );

q3 = cos( lati\_city\_1 + lati\_city\_2 );

dist = fix ( RRR \* acos( 0.5\*((1.0+q1)\*q2 - (1.0-q1)\*q3) ) + 1.0);

### Distance\_24

% Function distance\_24

% Evaluates de round trip distance

% Modified by PMO for evaluating geographical distances

function d = distance\_24(inputcities)

d = 0;

for n = 1 : length(inputcities)

if n == length(inputcities)

d = d + geo\_distance(inputcities(:,n),inputcities(:,1));

else

d = d + geo\_distance(inputcities(:,n),inputcities(:,n+1));

end

end