UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

INTELIGÊNCIA ARTIFICIAL
2024/2025

Professor

Paulo Oliveira

Eduardo Pires

RELATÓRIO

MÉTODOS DE PESQUISA

TRABALHO PRATICO 2 (PARTE A)

79881 - David Fidalgo
78800 - Tiago Carvalho

2024/2025

Índice

Índice de Figuras

1. Resumo

Este relatório documenta a implementação, análise e avaliação de dois métodos de otimização de pesquisa – Hill Climbing e o Simulated Annealing -, aplicados a um problema de uma função objetivo. Esta etapa foi realizada no MATLAB, incluindo o desenvolvimento dos algoritmos e a sua aplicação ao Problema do Caixeiro Viajante com alguns conjuntos de cidades de diferentes dimensões. Os testes realizados confirmaram a relevância de estratégias adaptativas na resolução de problemas de otimização, mostrando como diferentes ajustes nos algoritmos podem influenciar significativamente os resultados. Este relatório conclui com uma comparação entre todos os algoritmos, comparando os seus desempenhos.

2. Introdução

A busca por soluções ótimas em problemas de otimização é uma área fundamental da Inteligência Artificial, especialmente em cenários onde o espaço de busca é vasto e repleto de máximos locais. Este relatório aborda dois métodos heurísticos amplamente utilizados: o Hill Climbing e o Simulated Annealing, implementados e testados no ambiente MATLAB. O algoritmo Simulated Annealing foi aplicado tanto a funções matemáticas simples quanto ao Problema do Caixeiro Viajante (PCV), permitindo uma avaliação abrangente de seu desempenho.

Para além de implementar e comparar os métodos, o relatório inclui uma análise detalhada de seus resultados e a criação de uma tabela que identifica os pontos fortes e fracos de cada algoritmo. Esse recurso tem como objetivo sintetizar as conclusões do estudo e destacar as características que tornam cada método mais adequado para diferentes tipos de problemas de otimização.

Com isso, o trabalho não apenas explora a eficácia das abordagens implementadas, mas também propõe reflexões sobre suas aplicações práticas e o potencial de melhorias futuras, fornecendo uma visão completa e comparativa entre os dois algoritmos.

3. Enquadramento Teórico

3.1. Algoritmo da Subida da Colina

O algoritmo da subida da colina (ou Hill Climbing) é um método heurístico de pesquisa usado em problemas de otimização matemática na Inteligência Artificial. O objetivo passa por melhorar continuamente uma única solução por meio de pequenas alterações em cada iteração. Em cada passo, o algoritmo verifica se a nova solução é melhor que a anterior (Russell et al., 2020).

O processo começa com uma solução inicial, que é aperfeiçoada a cada iteração. Cada mudança é avaliada por uma função heurística que determina a qualidade da solução. O algoritmo continua até encontrar um máximo local, ou seja, um ponto em que não haja mais melhorias possíveis (Rao, 2009). Este algoritmo também é conhecido como um Algoritmo Ganancioso (ou Greedy Local Search) devido ao fato de escolher sempre as soluções que parecem ser as melhores no momento, sem considerar o impacto futuro dessas escolhas.

Tipos de Hill Climbing Algorithm: • Hill Climbing Simple – Faz alterações na solução atual e aceita a alteração, mas pode ficar preso em máximos locais. • Hill Climbing Multiple Restart – Faz múltiplas reinicializações em busca de uma solução melhor, aumentando a probabilidade de encontrar o máximo global (Ariful, 2015).

Tipos de Hill Climbing Algorithm (Ariful, 2015):

- Hill Climbing Simple – Faz alterações na solução atual e aceita a alteração, mas pode ficar preso em máximos locais.
- Hill Climbing Multiple Restart – Faz múltiplas reinicializações em busca de uma solução melhor, aumentado a probabilidade de encontrar o máximo global.

Variantes a ter em conta no Hill Climbing Algorithm:

- Solução inicial – Indica o ponto de partida da busca, escolhida de uma forma aleatória ou baseado em critérios heurísticos;
- Função de Avaliação – É o que avalia se uma solução está mais próxima de uma solução “ótima”;
- Vizinhança – Conjunto de soluções próximas à solução atual que podem ser alcançadas através de pequenas alterações;

- Critério de Aceitação – Define quando uma solução vizinha deve ser aceita. Em um problema de maximização, é aceite o vizinho se tiver a solução maior que a solução atual;
- Número de Iterações – Número de vezes que o algoritmo examina as soluções vizinhas;

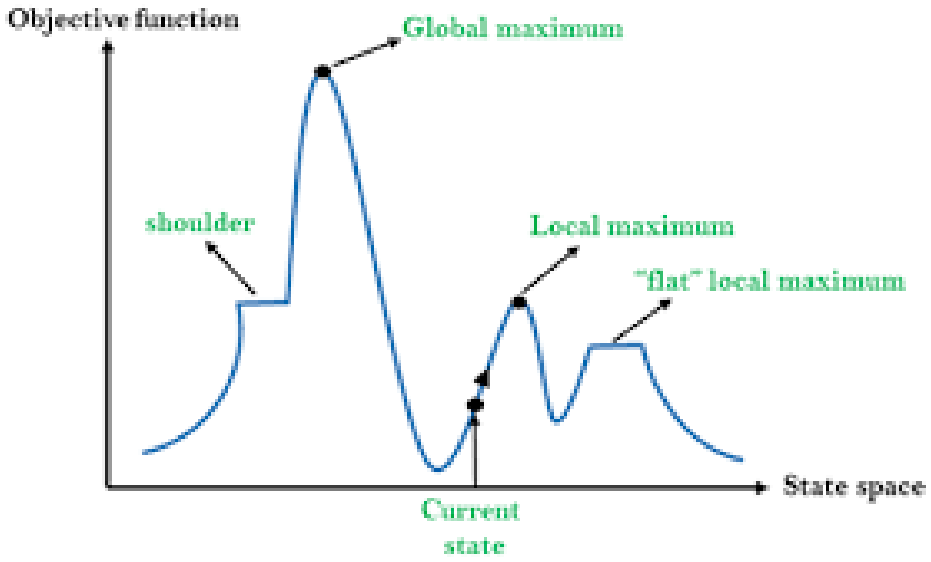

Figura 1. Hill Climb Algorithm: Gráfico exemplo

Explicação passo-a-passo:

- Inicialmente, é gerado uma solução aleatória ou então uma solução escolhida de forma heurística;
- Inicia um ciclo que será executado até que a condição imposta seja contrariada;
- Escolhe um dos vizinhos da solução na posição atual de forma aleatória;
- Verifica se o valor do vizinho é maior, menor ou igual à solução atual. Se tivermos perante um problema de maximização, o algoritmo vai verificar se o valor é maior que a solução atual, aceitando o vizinho se a condição se verificar;

3.2. Algoritmo Simulated Annealing

O **Algoritmo Simulated Annealing** (SA) é um método heurístico inspirado no processo de arrefecimento de metais. Na otimização, este método tenta encontrar soluções globais, evitando que o algoritmo fique preso em máximos/mínimos locais.

Diferente do **Hill Climbing Algorithm**, o **Simulated Annealing** poderá aceitar soluções “piores” durante a sua execução, permitindo assim uma exploração mais ampla do espaço de pesquisa. "Embora a simplicidade muitas vezes leve a soluções elegantes, a complexidade pode ser necessária para capturar a totalidade de um problema." (Knuth, D. 1997).

Variantes a ter em conta no Simulated Annealing:

- Solução Atual – Solução corrente na busca;
- Função de Custo – Mede a qualidade da solução;
- Temperatura – Controla a probabilidade de aceitar soluções piores;
- Fator de Arrefecimento (FA, 0<FA<1) – Taxa usada para reduzir a temperatura (T_new=T_atual*FA);
- Vizinhança – Soluções próximas à atual, geradas por ligeiras alterações;
- Diferença de Energia (ΔE) – Diferença entre o valor do custo da solução vizinha e da solução atual;
- Critério de Paragem – Número máximo de iterações ou temperatura mínima atingida.

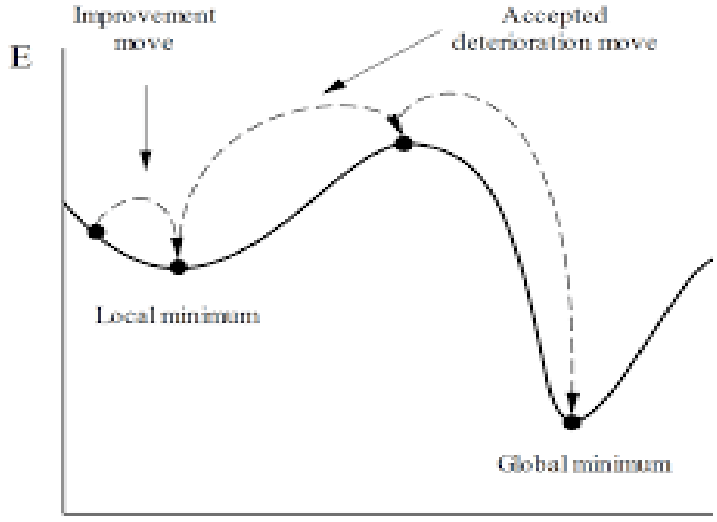

Figura 2-> Simulated Annealing Algorithm: Gráfico Exemplo

Explicação passo-a-passo:

- É gerado uma solução inicial aleatória ou heurística;
- É definida uma Temperatura inicial elevada para aceitar as soluções piores inicialmente;
- É gerado uma nova solução vizinha, realizando uma pequena perturbação na solução atual;
- Critério de aceitação:
    - Se a solução vizinha é melhor, ela é aceita;
    - Se é pior, poderá ser aceita pela seguinte função probabilística:

P=e^-(ΔE/T_atual)

- Arrefecimento – Reduz a temperatura de acordo com a taxa aplicada para o efeito;
- Critério de paragem – O SA para quando a temperatura atingiu um valor próximo de zero ou um número máximo de iterações é atingido.

4. Implementação dos Algoritmos

4.1. Enquadramento do Problema

Nesta fase de evolução deste trabalho prático, é proposto encontrar o máximo global da seguinte função:

$$f_1(x)=4(\sin(5\ \pi\ x+0.5)^6)\ exp\ (log_2\ ((x-0.8)^2))\quad 0\leq x\leq 1.6$$

Figura 3-> Função Unidimensional com intervalo especificado

Para resolver o problema numa fase inicial, foi utilizado o algoritmo *Hill Climb*, que começa com a geração de um ponto aleatório como solução inicial. A partir desse ponto, gera-se um novo ponto na vizinhança - se o novo ponto apresentar uma melhoria, ele substitui o ponto anterior como a melhor solução encontrada; caso contrário, mantém-se a solução atual, repetindo o processo até alcançar um ponto ótimo local.

Posteriormente, implementou-se a variante *Hill Climb* com reinicialização múltipla (*Multiple Restart Hill Climb*). Nesse método, após atingir um ótimo local, o algoritmo é reiniciado várias vezes a partir de diferentes pontos aleatórios, o que permite explorar outras regiões do espaço de soluções.

4.2. Resultados do Hill Climb

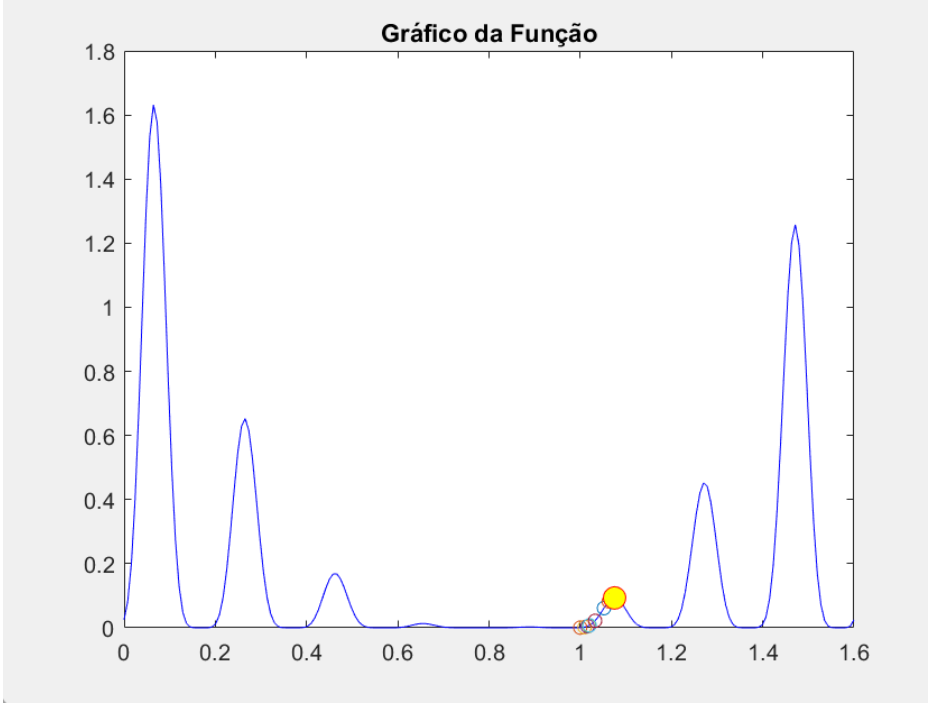

Figura 4-> Hill Climb: Gráfico da Função

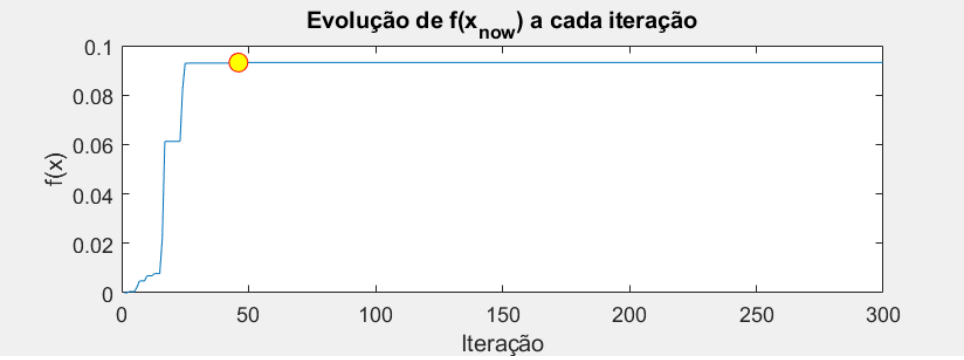

Figura 5-> Hill Climb: Evolução da solução

Figura 6-> Hill Climb: Evolução dos pontos

A **Figura 4** apresenta o gráfico da função objetivo, onde o ponto destacado corresponde a um máximo local. Este resultado evidencia a principal limitação do algoritmo **Hill Climb**: embora seja eficaz na busca por soluções "ótimas" dentro de uma vizinhança, é suscetível a ficar preso em máximos locais. Este comportamento exemplifica a sua natureza gananciosa, pois o algoritmo avança apenas para soluções que melhoram diretamente a função, sem considerar o espaço global de busca.

A **Figura 5** mostra a evolução do valor da função f(x) ao longo das iterações. Observa-se que o algoritmo faz melhorias contínuas até atingir o máximo local, representado pelo ponto destacado. Após alcançar este valor, não há mais progressão, reforçando a tendência do Hill Climb em convergir rapidamente para máximos locais.

Para melhorar o desempenho e aumentar as chances de encontrar o máximo global, aplica-se a técnica de **reinicialização múltipla** (*Multiple Restart*). Esta estratégia consiste em reiniciar o algoritmo várias vezes a partir de diferentes pontos aleatórios no espaço de busca, permitindo uma exploração mais ampla e diversificada. Ao adotar esta abordagem, o algoritmo tem maior probabilidade de escapar de máximos locais e identificar a melhor solução global.

4.3. Resultados do Hill Climb Multiple Restart

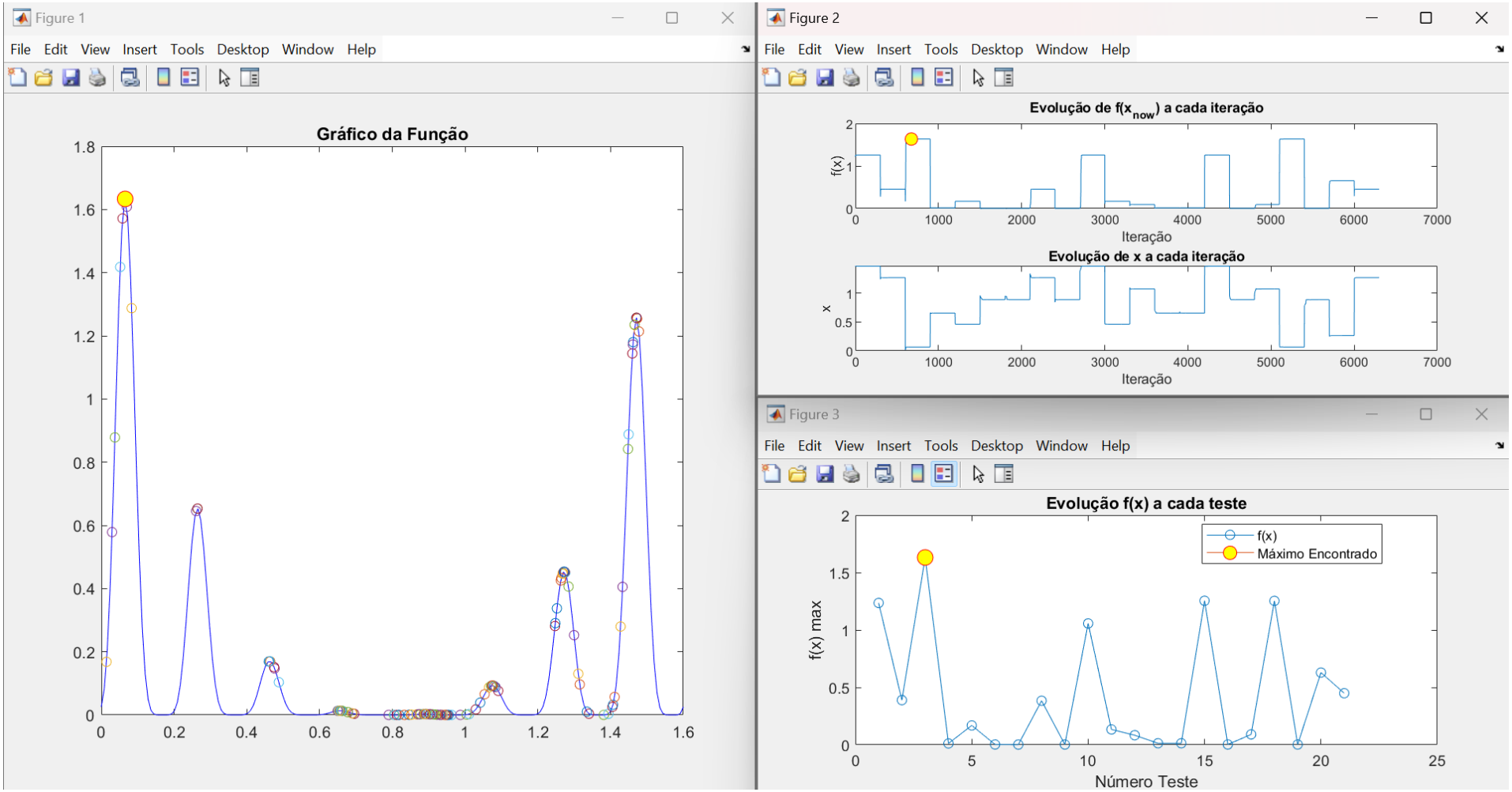

Figura 7-> Hill Climb Multiple Restart: Resultado

No teste realizado, verifica-se que o máximo global foi encontrado no final do 3.º teste. A estratégia utilizada incrementa 10 reinicializações sempre que é encontrada uma nova melhor solução. Desta forma, o processo totalizou 21 reinicializações: inicialmente começa com uma única reinicialização e, a cada nova solução ótima encontrada, são adicionadas mais 10 reinicializações, forçando o algoritmo a procurar o máximo global de forma mais exaustiva e eficaz.

Analisando os gráficos:

- Gráfico da Função Objetivo -> Observa-se a função objetivo e os pontos explorados pelo algoritmo. O ponto assinalado em amarelo indica o máximo global encontrado. Os restantes pontos mostram a exploração do espaço de busca.

- Evolução ao Longo das Iterações -> No gráfico superior, a evolução do f(x) em todos os testes indica momentos em que o algoritmo encontrou melhores soluções e as manteve até uma nova melhoria. O gráfico inferior ilustra as mudanças abruptas nos valores de x, evidenciando as reinicializações frequentes, características do método do Multiple Restart.

- Evolução f(x) a Cada Teste -> O máximo global foi identificado no 3.º teste, conforme destacado. As oscilações nos valores máximos encontrados em cada teste refletem a exploração do espaço. Após o 3.º teste, as reinicializações adicionais, até ao 21.º teste, não encontraram soluções superiores, mas reforçaram a robustez da procura ao garantir uma exploração mais abrangente.

4.3.1. Hill-Climbing VS Hill-Climbing Multiple Restart

Os algoritmos Hill Climb e Hill Climb Multiple Restart partilham a mesma abordagem fundamental de melhoria iterativa de soluções, mas diferem significativamente na capacidade de escapar de máximos locais e alcançar o máximo global.

Hill Climb:

Conforme observado nas Figuras 4, 5 e 6, o Hill Climb apresenta um comportamento ganancioso, ou seja, avança apenas para soluções que melhoram diretamente o valor da função objetivo f(x)f(x)f(x). Este método mostrou-se eficaz em encontrar soluções localmente ótimas rapidamente. No entanto, uma limitação evidente é a sua suscetibilidade a ficar preso em máximos locais, como ilustrado pelo ponto destacado nos gráficos. Assim que o algoritmo atinge um máximo local, não há mais progressão, uma vez que não há mecanismos intrínsecos para escapar dessas soluções sub-ótimas.

Hill Climb Multiple Restart:

Em contraste, o Hill Climb Multiple Restart, cujos resultados estão refletidos na Figura 7, utiliza uma estratégia de reinicializações sistemáticas para mitigar o problema dos máximos locais. A cada vez que o algoritmo encontra uma solução superior, adiciona mais reinicializações, o que força uma busca mais exaustiva no espaço de soluções. Este método mostrou-se mais eficaz ao encontrar o máximo global no 3.º teste, continuando com mais 18 reinicializações adicionais para garantir a robustez da solução.

Comparação:

Capacidade de Exploração:

O Hill Climb realiza uma exploração local limitada e rápida, mas com maior risco de estagnação em máximos locais.

O Hill Climb Multiple Restart expande significativamente a exploração ao permitir múltiplas tentativas a partir de diferentes pontos iniciais, aumentando a probabilidade de alcançar o máximo global.

Robustez e Eficácia:

O Hill Climb é eficiente para problemas onde o máximo global está próximo do ponto inicial ou onde o espaço de busca não possui muitos máximos locais.

O Hill Climb Multiple Restart, embora mais intensivo em termos de iterações, é mais robusto, garantindo uma busca mais abrangente e eficaz, como demonstrado pelo sucesso em encontrar o máximo global nos testes apresentados.

Natureza do Algoritmo:

O Hill Climb é um método ganancioso, sempre focado em melhorias imediatas.

O Hill Climb Multiple Restart adota uma abordagem híbrida, combinando exploração local com reinicializações estratégicas, superando a natureza gananciosa do Hill Climb simples.

Em suma

O Hill Climb Multiple Restart apresenta-se como uma versão mais eficaz e robusta do Hill Climb, especialmente em cenários onde o espaço de busca é complexo e contém múltiplos máximos locais. A estratégia de reinicializações permite ao algoritmo escapar de soluções sub-ótimas e encontrar o máximo global, como demonstrado nos resultados em cima apresentados.

4.4. Resultados do Simulated Annealing (SA)

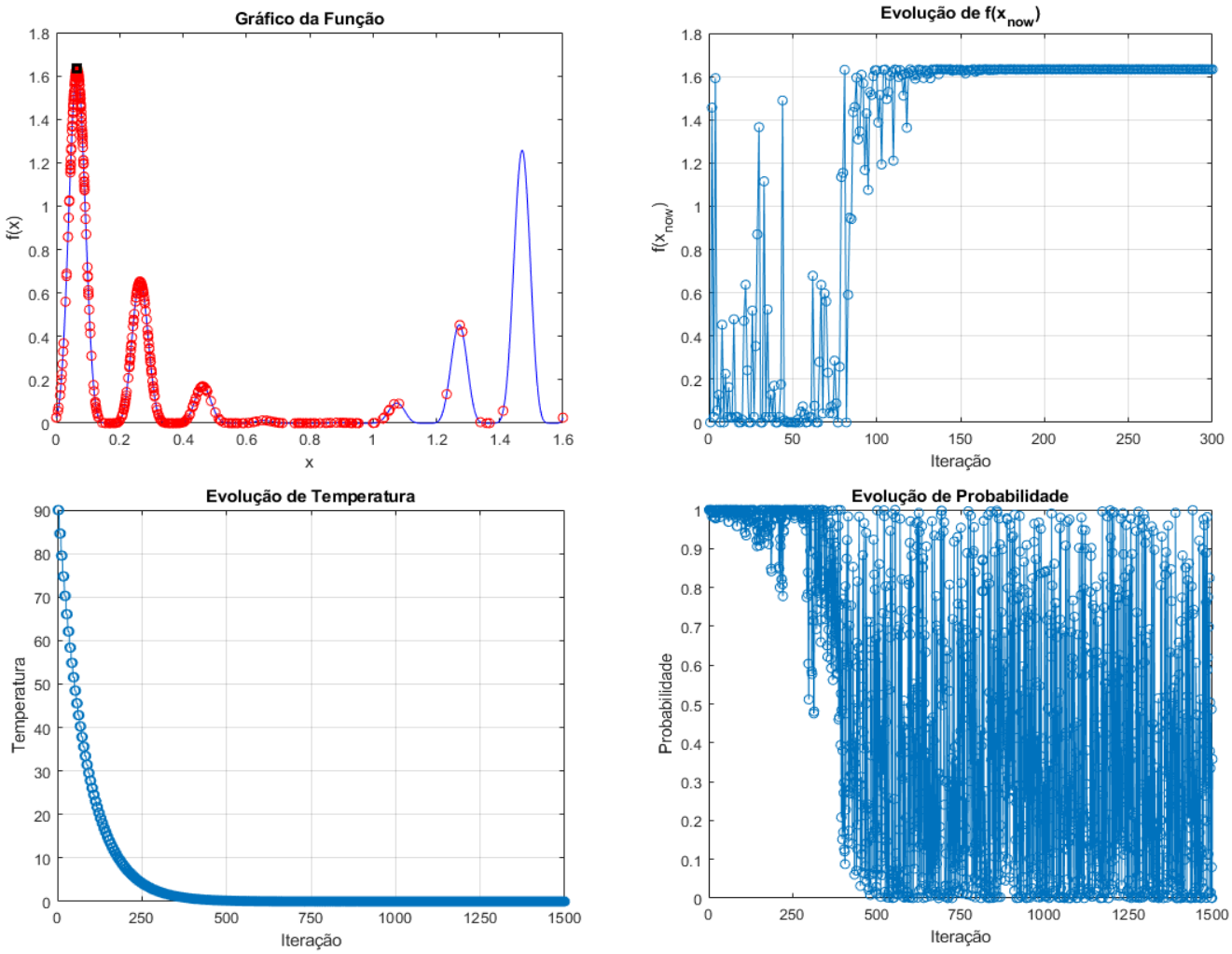

Figura 8-> Simulated Annealing

Nesta etapa do trabalho, o algoritmo Simulated Annealing (SA) foi implementado e testado para encontrar o máximo global da função unidimensional presente no **Gráfico da Função** da Figura 9. Este método heurístico, que é inspirado no processo de arrefecimento lento em metalurgia, procura encontrar soluções globais mesmo em espaços de busca com múltiplos máximos locais. Por isso é que aceita soluções "piores" temporariamente na tentativa de evitar convergir para esses mesmos espaços.

4.4.1. Análise dos Resultados Gráficos (Figura 9):

Gráfico da Função: Representada aqui a aplicação do SA resultou da exploração ampla do espaço de busca da função objetivo, como vemos pela distribuição dos pontos vermelhos no gráfico. Estes pontos representam as soluções testadas ao longo das iterações. A trajetória do algoritmo, visível através da sequência de pontos vermelhos,

demonstra a capacidade do SA de escapar de máximos locais, explorando regiões com valores de função menores para evitar a convergência prematura. O quadrado preto destaca a solução final encontrada pelo algoritmo, a qual se aproxima significativamente do máximo global da função. A distribuição dos pontos vermelhos, especialmente nas regiões iniciais, onde há maior dispersão, ilustra a fase exploratória do SA, enquanto a concentração em torno do quadrado preto indica a convergência para uma solução ótima.

Evolução de f(x_{now}): A evolução do valor da função f(x_{now}) ao longo das iterações revela o comportamento característico do SA. Inicialmente, observam-se grandes flutuações no valor de f(x_{now}), indicando a aceitação de soluções piores durante a fase de alta temperatura. Estas flutuações são cruciais para a exploração do espaço de busca e para evitar que o algoritmo fique preso em máximos locais. À medida que as iterações avançam, a amplitude das flutuações diminui gradualmente, indicando a redução da temperatura e da probabilidade de aceitar soluções piores. Eventualmente, o algoritmo converge para um valor estável, próximo do máximo global, demonstrando a eficácia do SA em encontrar soluções ótimas mesmo em funções complexas.

Evolução da Temperatura: A figura ilustra a redução da temperatura ao longo do processo de otimização. A temperatura inicial, definida como 90, é crucial para permitir a aceitação de soluções piores e a exploração abrangente do espaço de busca. A diminuição exponencial da temperatura, é controlada pelo fator de arrefecimento (alfa = 0.94), reduz gradualmente a probabilidade de aceitar soluções piores, conduzindo o algoritmo à convergência. A curva suave e decrescente da temperatura evidencia o processo de "arrefecimento" análogo à metalurgia, onde a diminuição controlada da temperatura permite a formação de estruturas estáveis.

Evolução da Probabilidade: A evolução da probabilidade de aceitar uma solução pior, apresentada na figura, está diretamente relacionada à temperatura. No início, a alta temperatura resulta em uma alta probabilidade de aceitar soluções piores, o que facilita a exploração de diferentes regiões do espaço de busca. À medida que a temperatura diminui, a probabilidade de aceitar soluções piores também decresce, tornando o algoritmo mais seletivo e direcionando-o para a convergência. As flutuações na probabilidade refletem a natureza estocástica do SA, onde a aceitação de soluções piores é determinada por um sorteio aleatório, influenciado pela temperatura atual.

4.4.2. Hill Climbing VS Simulated Annealing

Comparado ao algoritmo Hill Climbing, o Simulated Annealing demonstrou ser mais robusto na busca do máximo global. Enquanto o Hill Climbing tende a convergir para o

máximo local mais próximo do ponto inicial, o SA, ao permitir a aceitação de soluções piores, consegue explorar uma área maior do espaço de busca e escapar de máximos locais. As oscilações observadas na Figura 2, especialmente nas iterações iniciais, evidenciam a capacidade do SA de explorar regiões menos promissoras para evitar a estagnação em soluções sub-ótimas.

4.4.3. Balanço final

O algoritmo Simulated Annealing provou ser uma ferramenta eficaz para encontrar o máximo global da função proposta. A estratégia de aceitar soluções piores, controlada pela temperatura e probabilidade, permitiu uma exploração abrangente do espaço de busca, resultando na convergência para uma solução próxima do ótimo global. Os resultados gráficos confirmam a capacidade do SA de evitar máximos locais e encontrar soluções de alta qualidade. Este estudo reforça a importância do Simulated Annealing como um método robusto para problemas de otimização complexos, especialmente aqueles com múltiplos máximos ou mínimos locais. A escolha adequada dos parâmetros, como a temperatura inicial e o fator de arrefecimento, é crucial para o desempenho do algoritmo e a obtenção de resultados satisfatórios.

4.6. Resultados do Problema do Caixeiro Viajante (PCV), usando o Simulated Annealing (SA)

O objetivo desta etapa foi aplicar o algoritmo do Simulated Annealing para o Problema do Caixeiro Viajante e avaliar os resultados obtidos ao longo das iterações. A análise dos próximos testes com os seus respetivos gráficos permite compreender a evolução do percurso e do custo total (distância percorrida) durante a execução do algoritmo.

Para a realização desta etapa, foi fornecido os seguintes conjunto de cidades para a realização deste algoritmo:

#	*Cidade*	*Latitude*	*Longitude*
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	Peso da Régua	41N10	7W47
12	Guimarães	41N27	8W18
13	Valença do Minho	42N02	8W38
14	Barcelos	41N32	8W37

Figura 9-> PCV: Conjunto 14 cidades

#	*Cidade*	*Latitude*	*Longitude*
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

Figura 10-> PCV: Conjunto 20 cidades

#	City	Latitude	Longitude	#	City	Latitude	Longitude
1	Bragança	41N49	6W45	16	Tavira	37N07	7W39
2	Vila Real	41N18	7W45	17	Sagres	37N00	8W56
3	Chaves	41N44	7W28	18	Setúbal	38N32	8W54
4	Viana do Castelo	41N42	8W50	19	Guarda	40N32	7W16
5	Braga	41N33	8W26	20	Santarém	39N14	8W41
6	Aveiro	40N38	8W39	21	Beja	38N01	7W52
7	Porto	41N11	8W36	22	Sines	37N57	8W52
8	Viseu	40N39	7W55	23	Covilhã	40N17	7W30
9	Lamego	41N06	7W49	24	Tomar	39N36	8W25
10	Guimarães	41N27	8W18	25	Águeda	40N34	8W27
11	Coimbra	40N12	8W25	26	Leiria	39N45	8W48
12	Faro	37N01	7W56	27	Castelo Branco	39N49	7W30
13	Évora	38N34	7W54	28	Elvas	38N53	7W10
14	Lisboa	38N43	9W10	29	Miranda do	41N30	6W16
15	Portalegre	39N17	7W26	30	Sintra	38N48	9W23

Figura 11-> PCV: Conjunto 30 cidades

4.6.1. PCV com 14 cidades

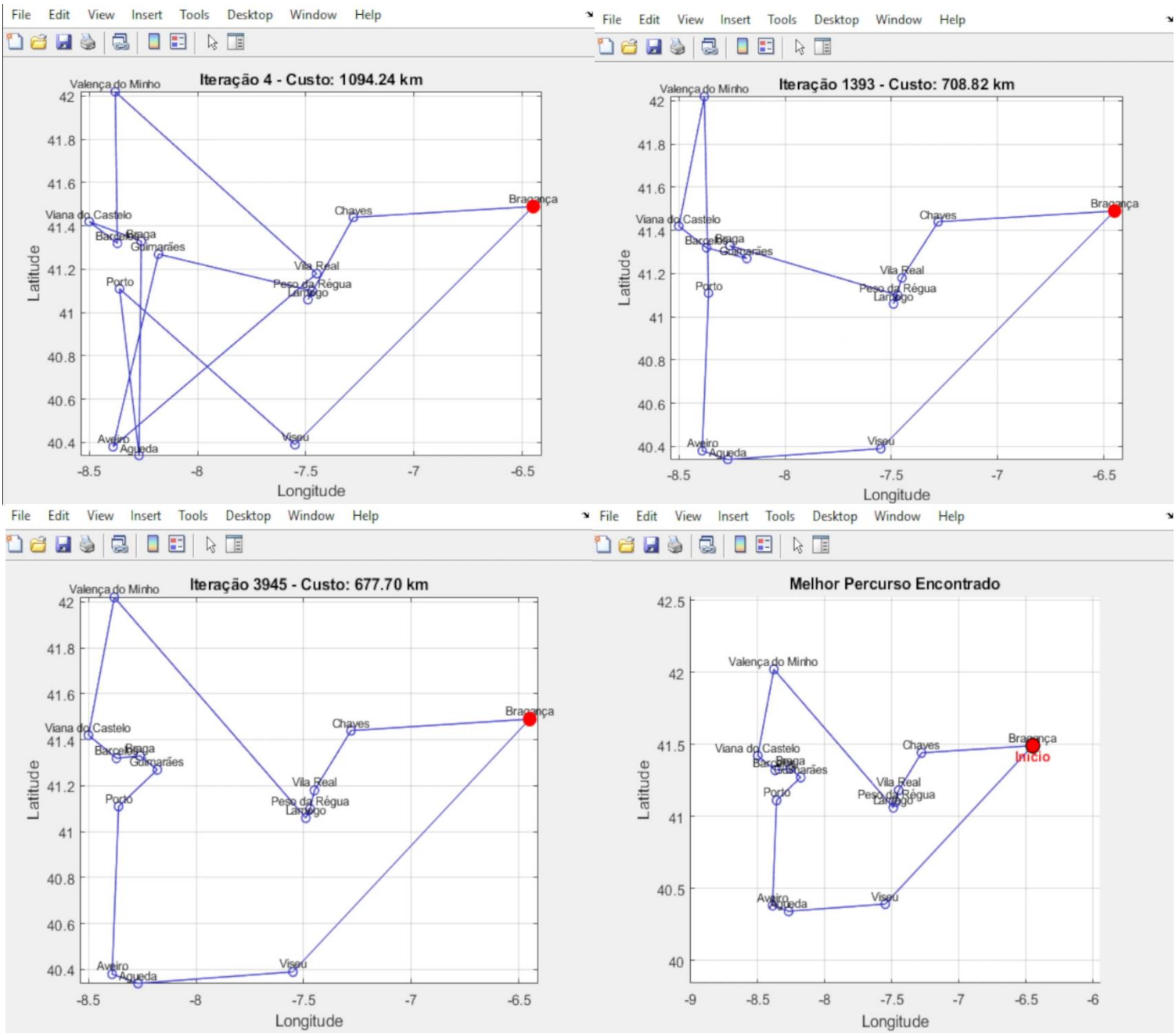

Figura 12-> 1º Teste PCV 14 cidades: Caminhos de iterações Aleatórias

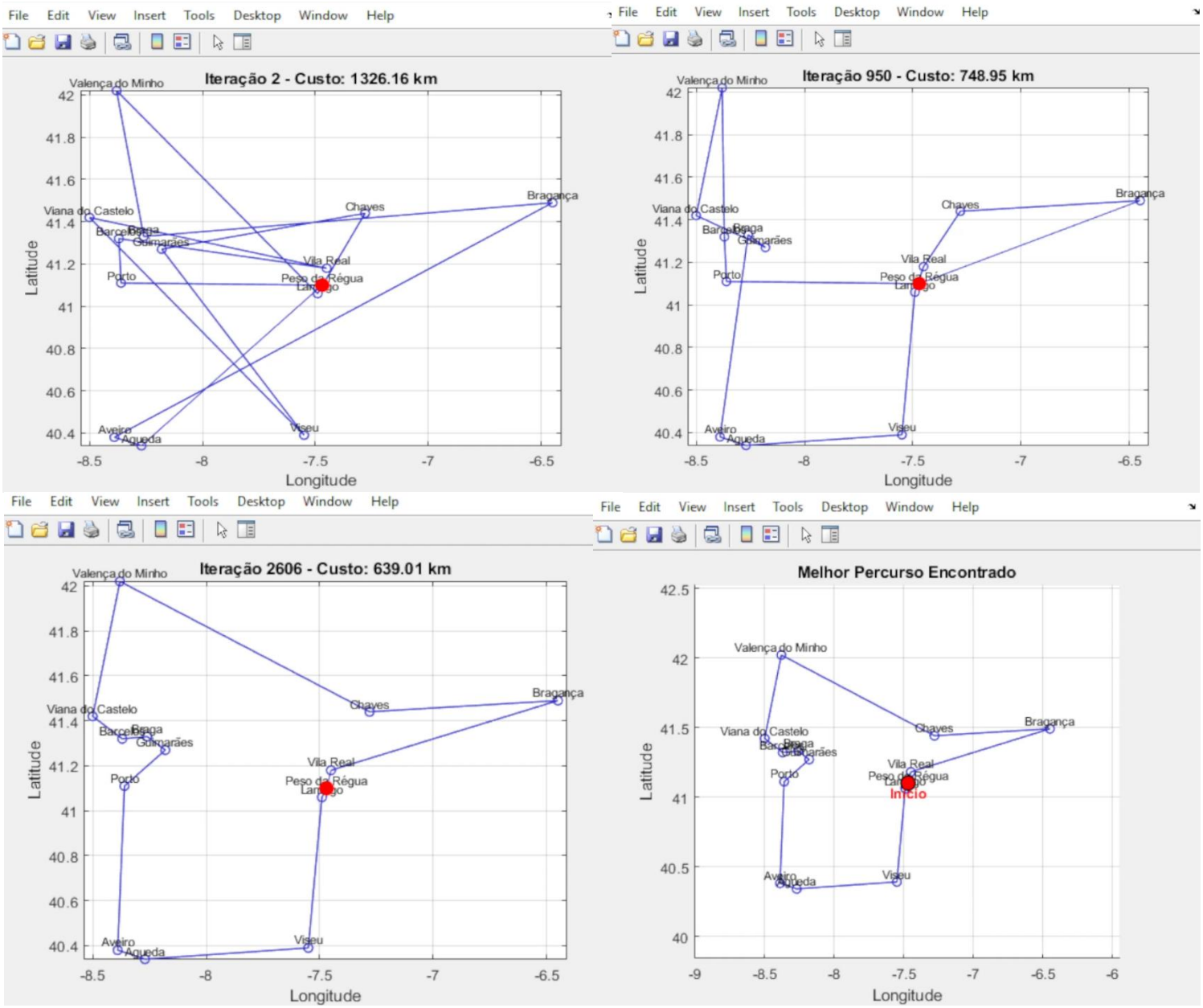

Figura 13-> 2º Teste PCV 14 cidades: Caminhos de iterações Aleatórias

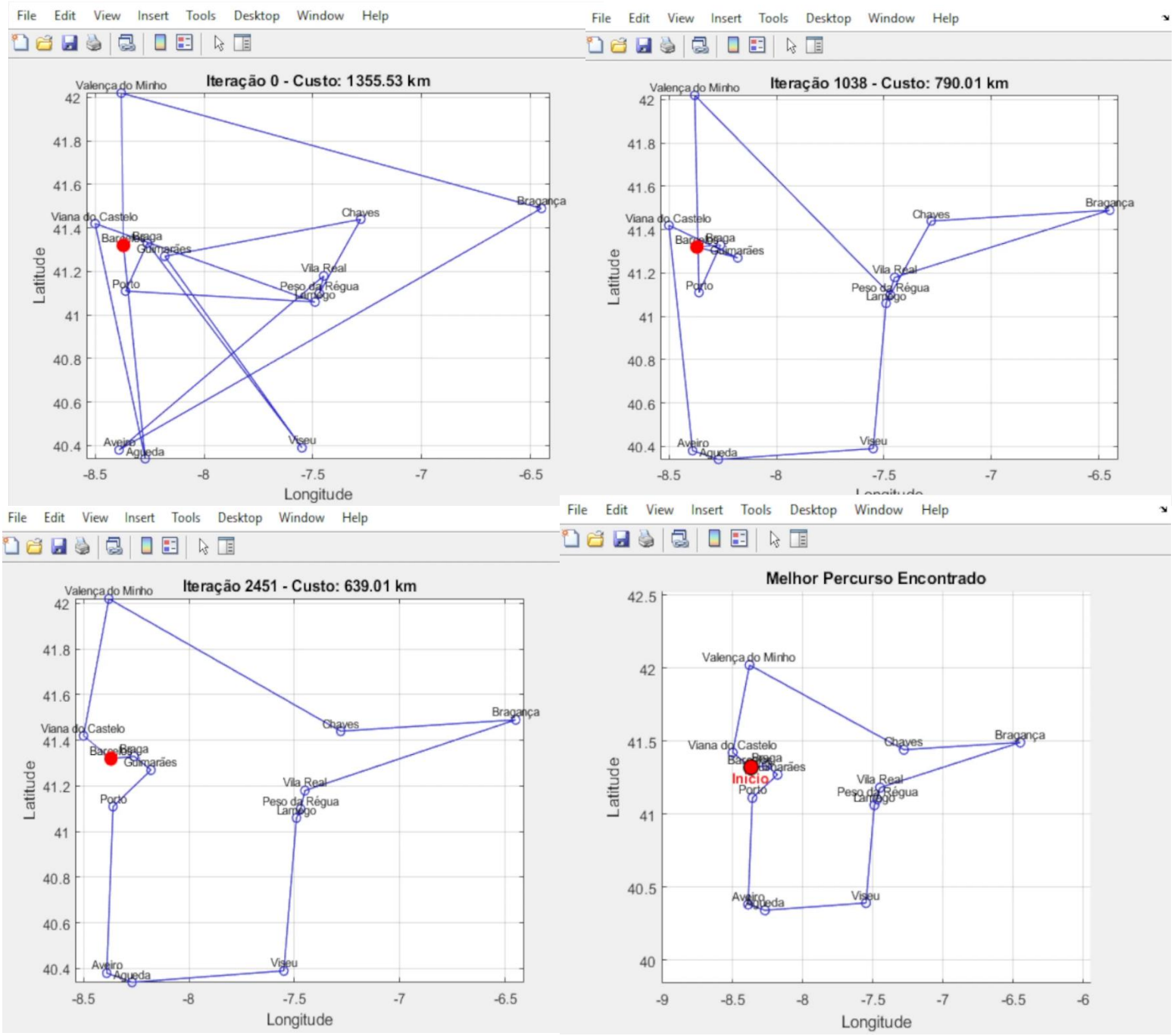

Figura 14-> 3º Teste PCV 14 cidades: Caminhos de iterações Aleatórias

Os resultados obtidos mostram a evolução clara do percurso ao longo das iterações do algoritmo SA aplicado ao Problema do Caixeiro Viajante.

Neste caso, existem 87.178.291.200 percursos diferentes (14!). Ou seja, encontrar um caminho mais curto exige, assim, uma adaptação leve das variáveis iniciais, que neste caso foram:

- Temperatura Inicial-> 90;
- Alpha->0.96;
- Iterações a cada valor de temperatura-> 50

Tendo em conta o 3º teste, que é o teste onde a diferença entre a primeira iteração e a última é significativa entre os testes realizados, o processo é gerado aleatoriamente, definindo também um ponto inicial (neste caso, a cidade de Barcelos), resultando num custo de 1355,53 Km.

Seguidamente, na iteração 1038, o custo encontra se reduzido para 790,01 Km, o que demonstra que o SA já realizou importantes ajustes, eliminando redundâncias e identificando combinações mais curta, fruto também da aceitação de custos “piores”, em busca do melhor algoritmo possível

Na iteração 2451, que equivale à última iteração que foi encontrado o melhor caminho, foi atingido o custo de 639,01 Km.

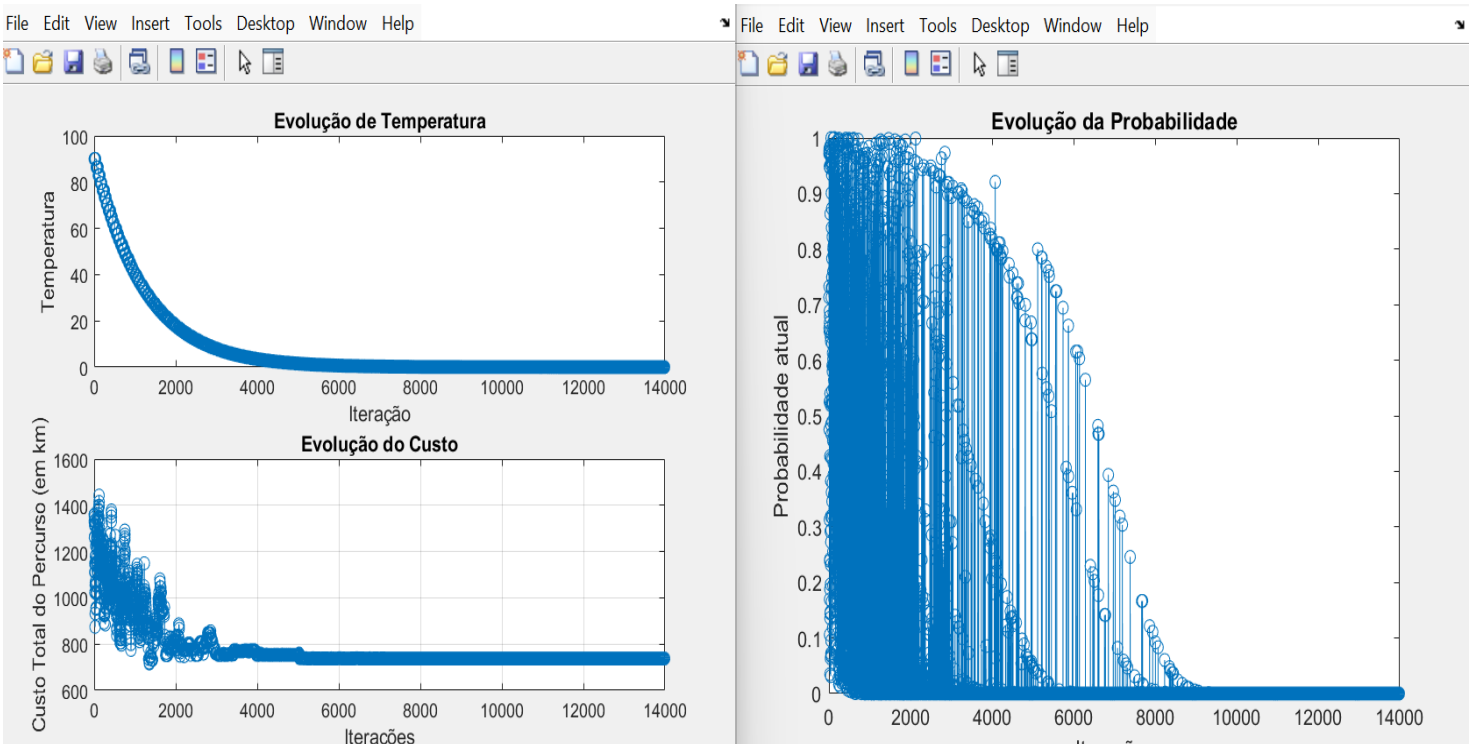

Figura 15 -> 3º Teste PCV 14 cidades: Gráficos de Estudo

Com isto, com estes resultados apresentados, foi encontrado o percurso com um custo “ótimo”, tal como pretendido.

4.6.2. PCV com 20 cidades

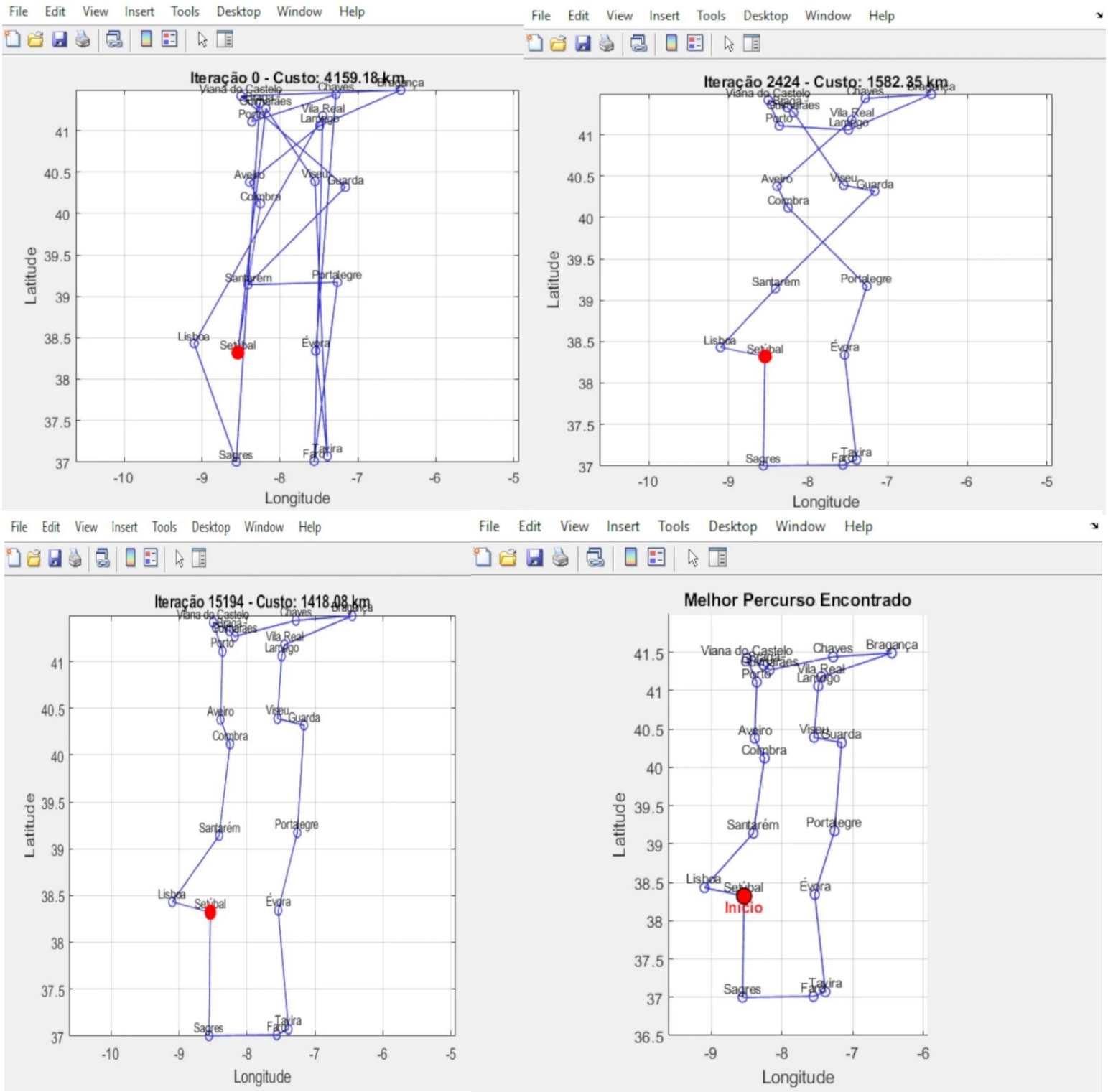

Figura 16-> 1º Teste PCV 20 cidades: Caminhos de iterações Aleatórias

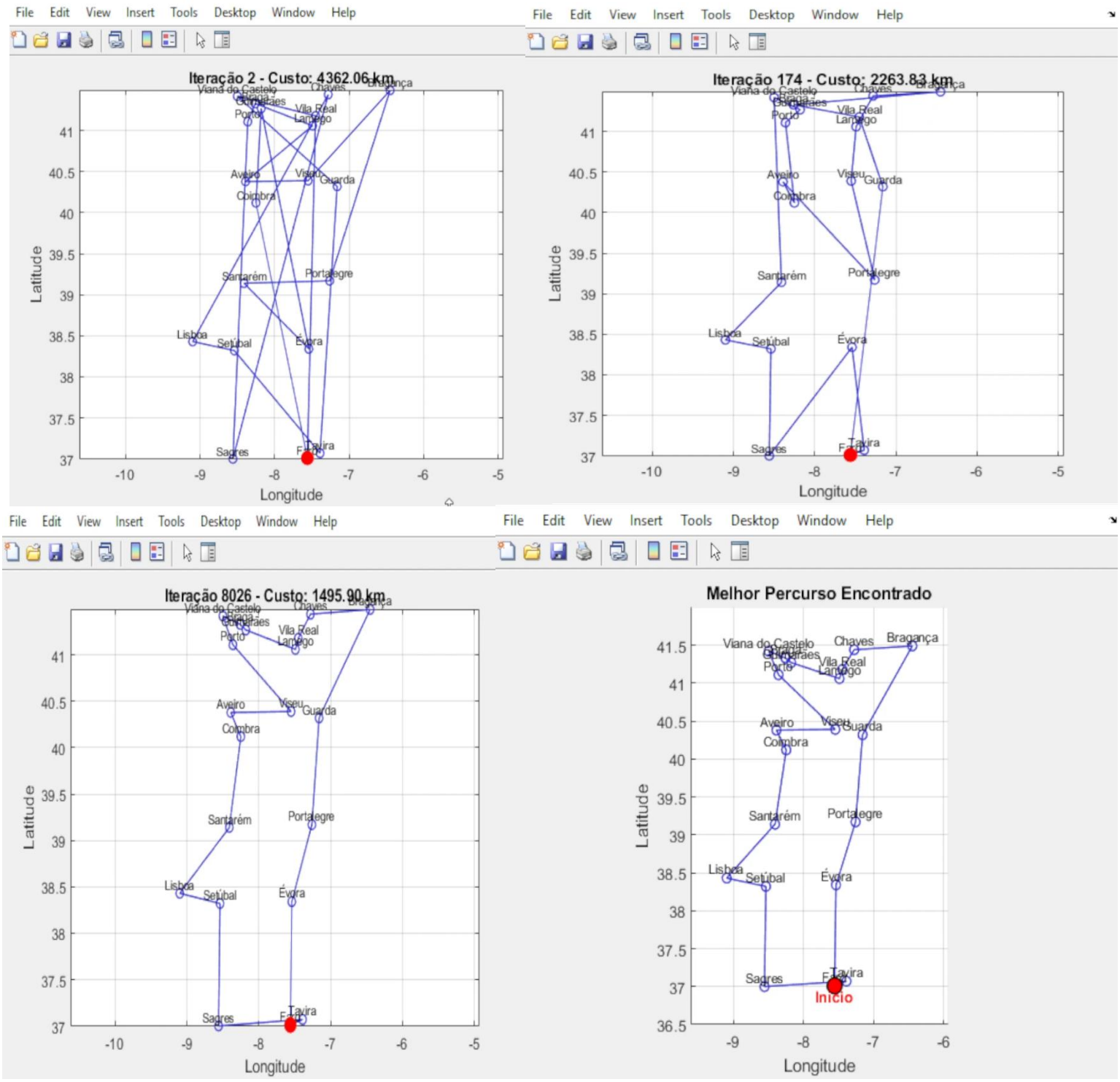

Figura 17-> 2º Teste PCV 20 cidades: Caminhos de iterações Aleatórias

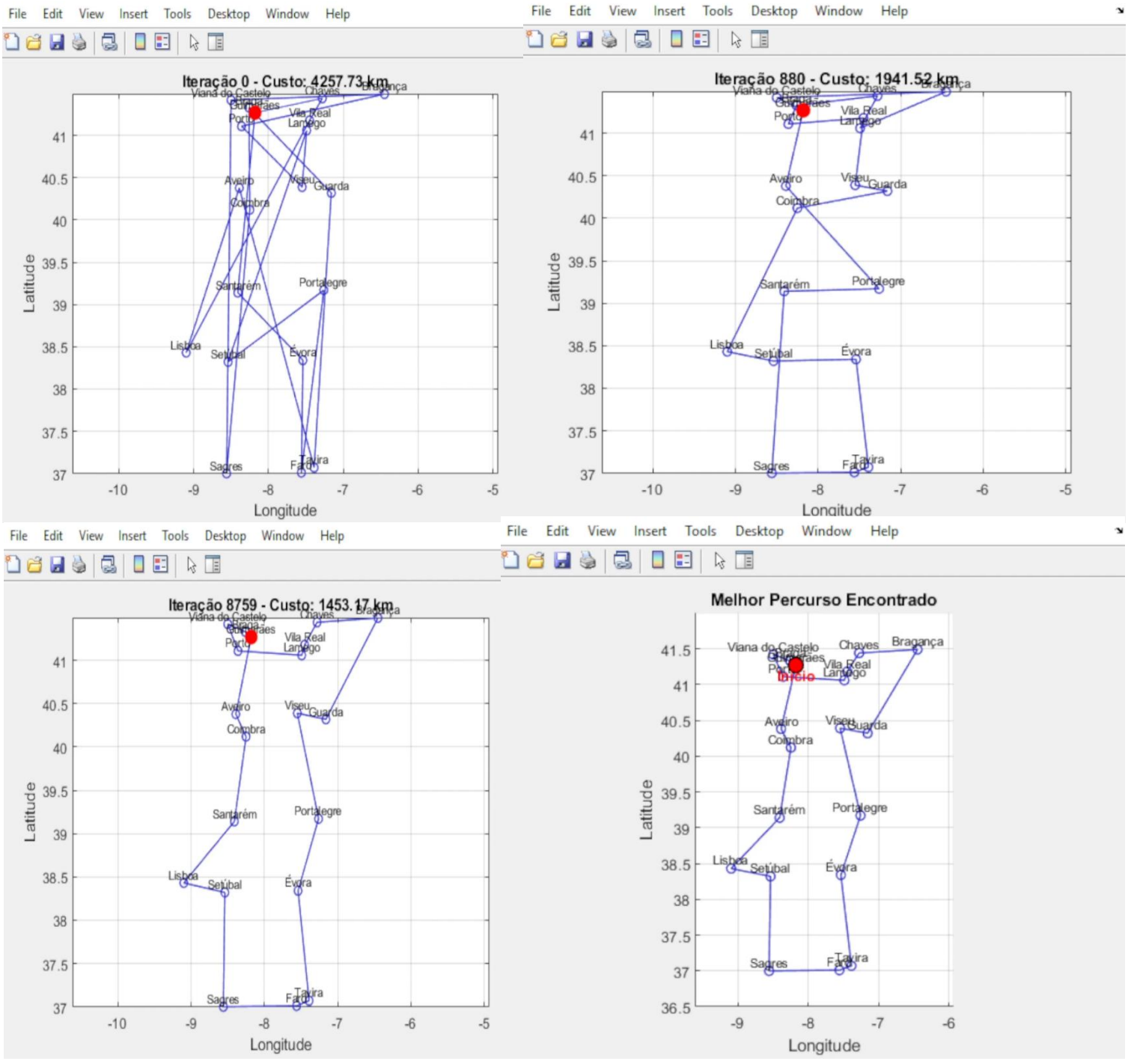

Figura 18-> 3º Teste PCV 20 cidades: Caminhos de iterações Aleatórias

Neste conjunto, cujo objetivo passa por encontrar o caminho de menor custo que envolve o norte e o sul de Portugal, existem 2.432.902.008.176.640.000 percursos diferentes (20!). Com o grau de dificuldade a aumentar, foi adaptado os valores iniciais das variáveis cruciais para este algoritmo:

- Temperatura Inicial-> 90;
- Alpha->0.98; (Para um arrefecimento Lento)
- Iterações a cada valor de temperatura-> 100

Tendo em conta o 1º teste, cujo resultado é o mais desejado, encontra-se que a solução inicial formada passa pelo valor do custo de 4159.18 Km.

Depois de várias alterações, na iteração 2424, o custo encontra se reduzido para 1582.35 Km. Na figura 19, é possível reparar que é entre as iterações 2000 e 5000 que o algoritmo encontra os melhores resultados

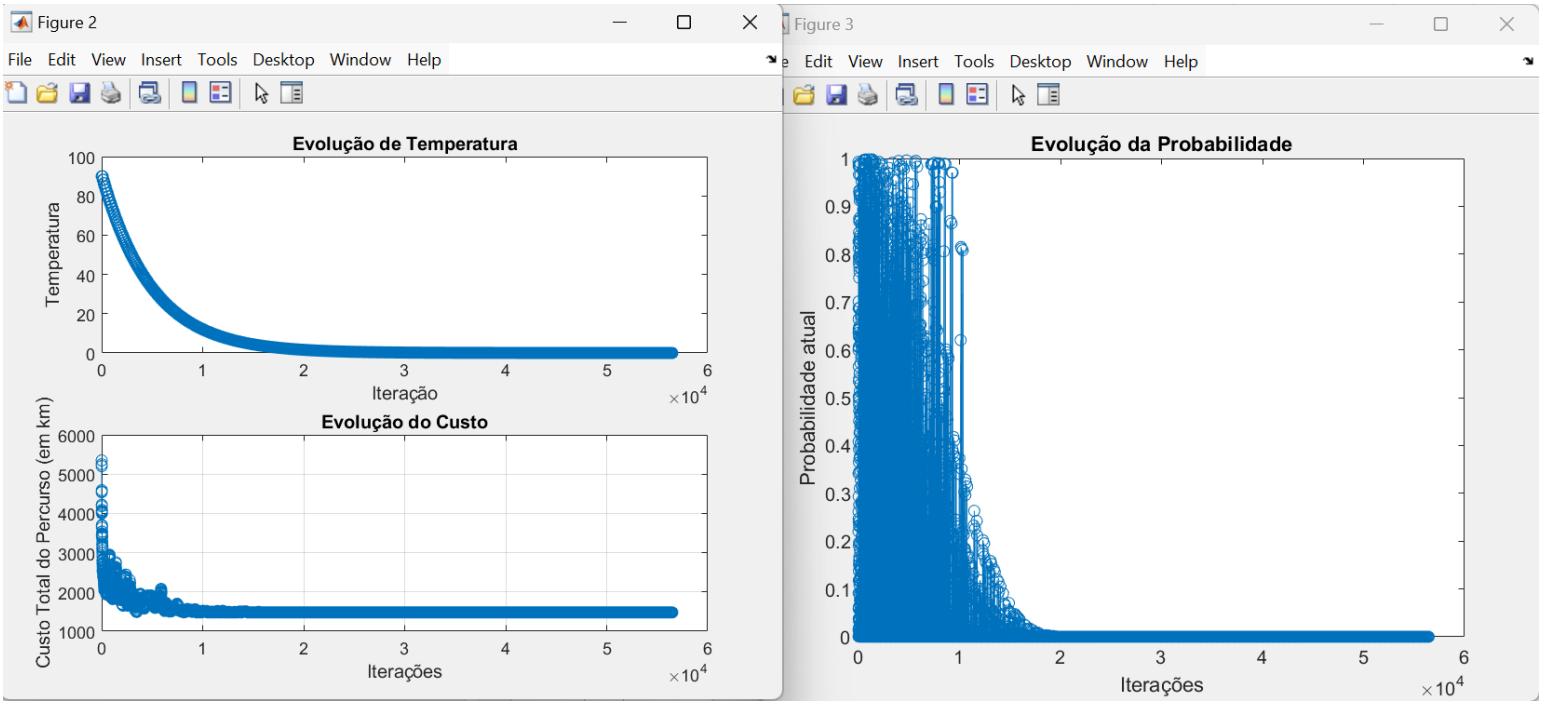

Figura 19-> 1º Teste PCV 20 cidades: Gráficos de Estudo

Na iteração 8759, que equivale à última iteração em que foi encontrado o melhor caminho, foi atingido o custo de 639,01 Km.

Nos 3 testes efetuados, o melhor custo apresentado é do da figura 16, fixando se de uma solução “ótima”.

4.6.3. PCV com 30 cidades

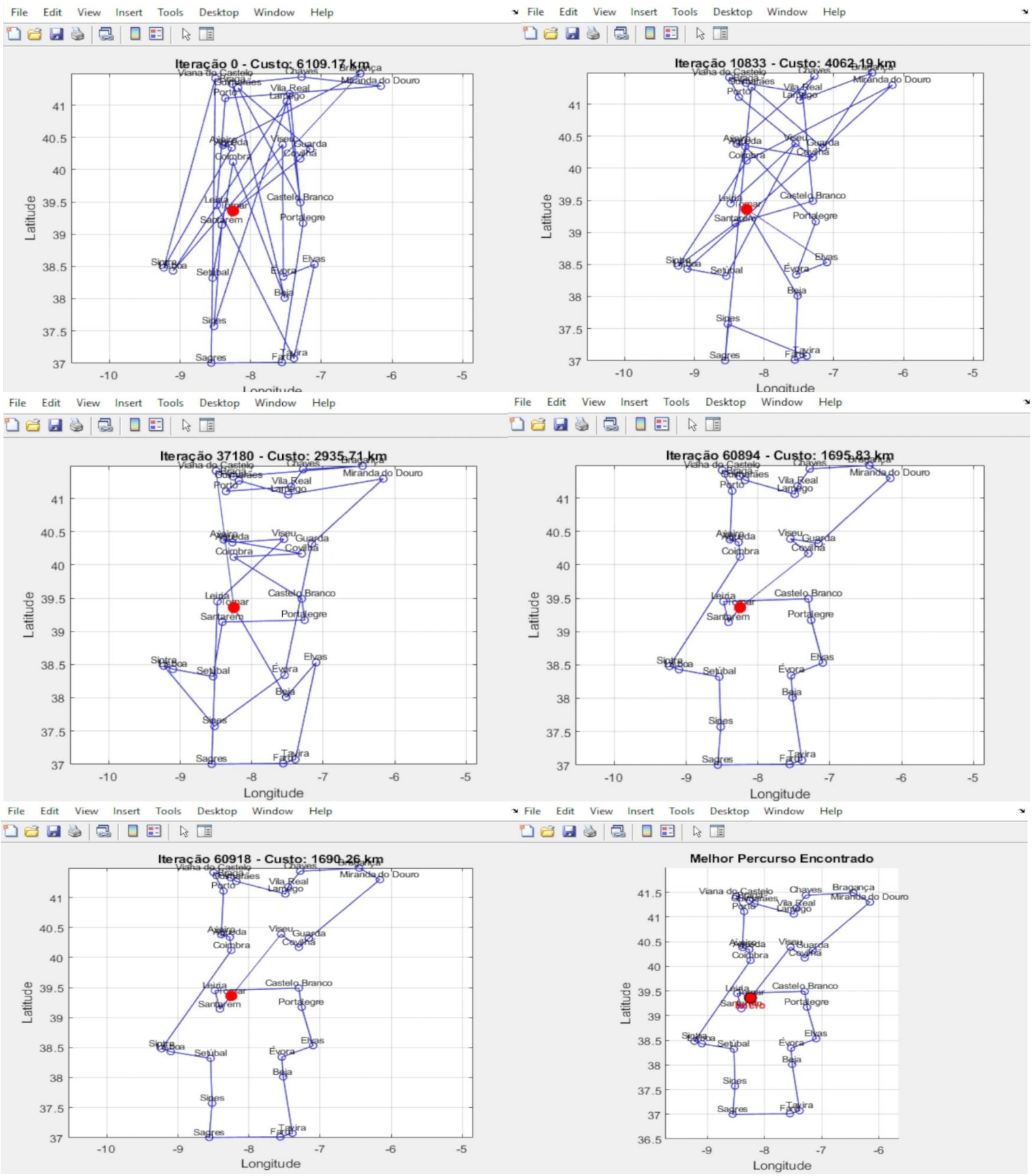

Figura 20-> 1º Teste PCV 30 cidades: Caminhos de iterações Aleatórias

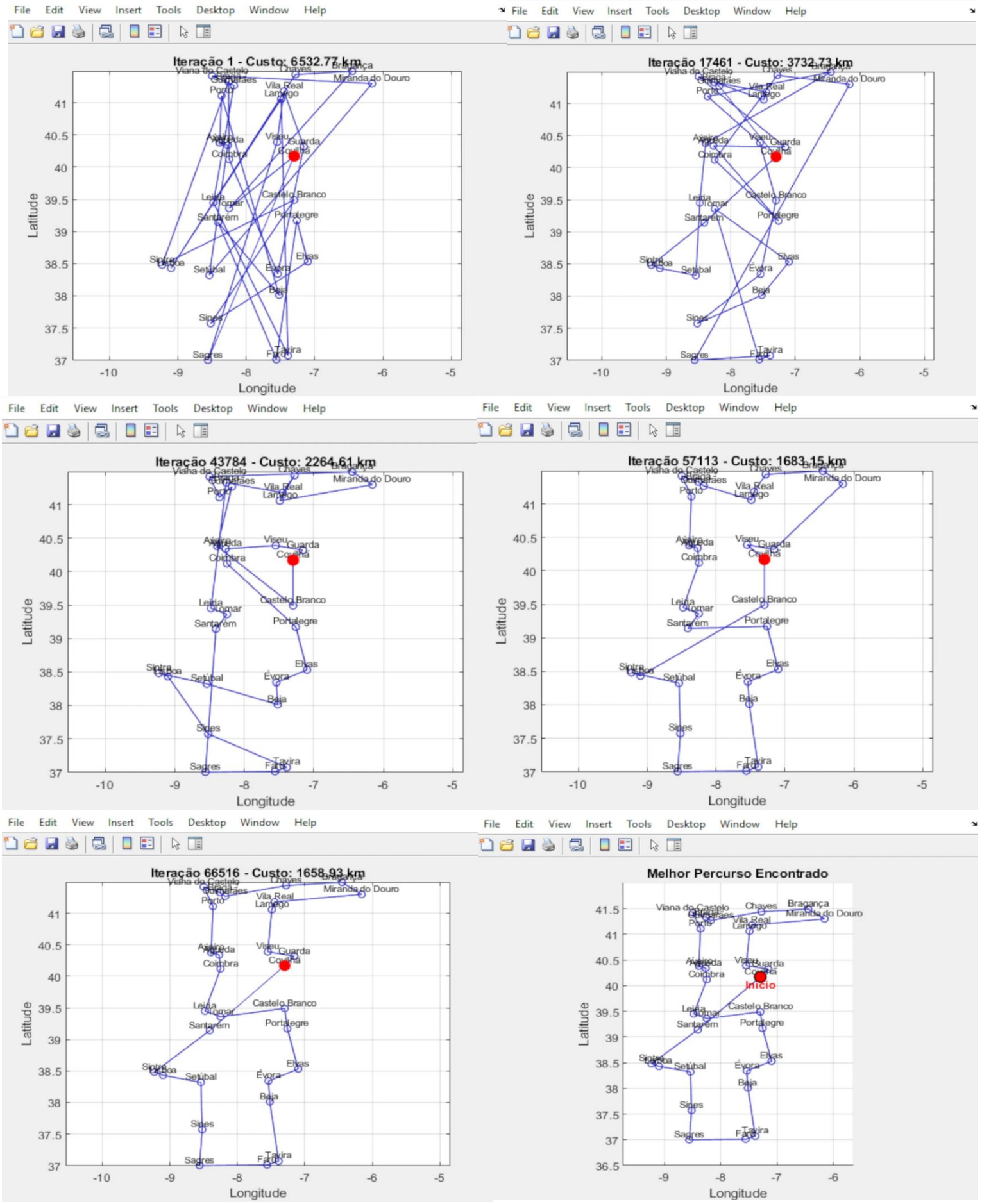

Figura 21-> 2º Teste PCV 30 cidades: Caminhos de iterações Aleatórias

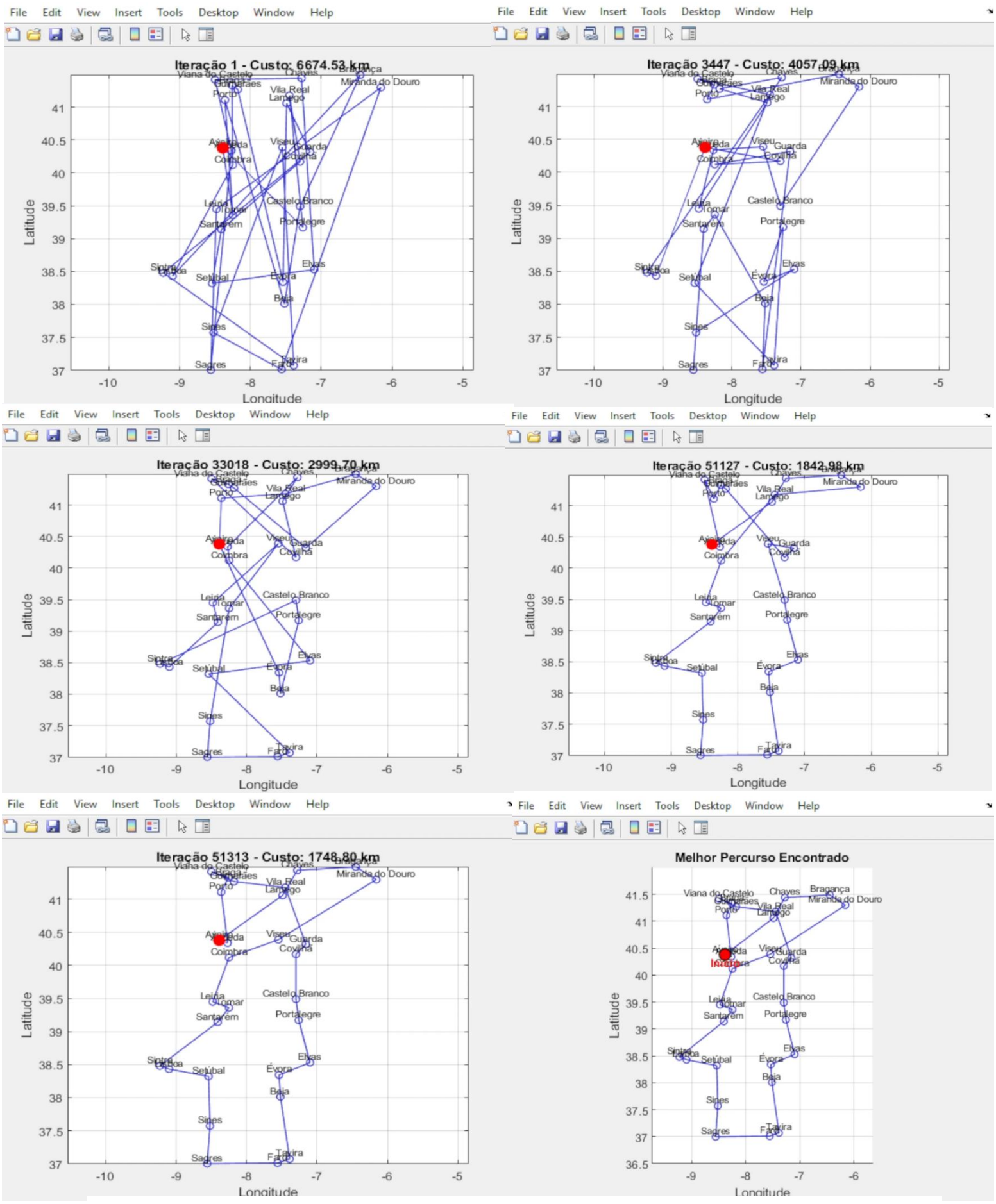

Figura 22-> 3º Teste PCV 30 cidades: Caminhos de iterações Aleatórias

Neste último conjunto, onde existem 2.6525286*10^{32} percursos diferentes (30!), o que obriga, em semelhança ao que aconteceu no conjunto anterior, adaptar os valores iniciais dos parâmetros cruciais para este algoritmo

- Temperatura Inicial-> 1000;
- Alpha->0.98;
- Iterações a cada valor de temperatura-> 300

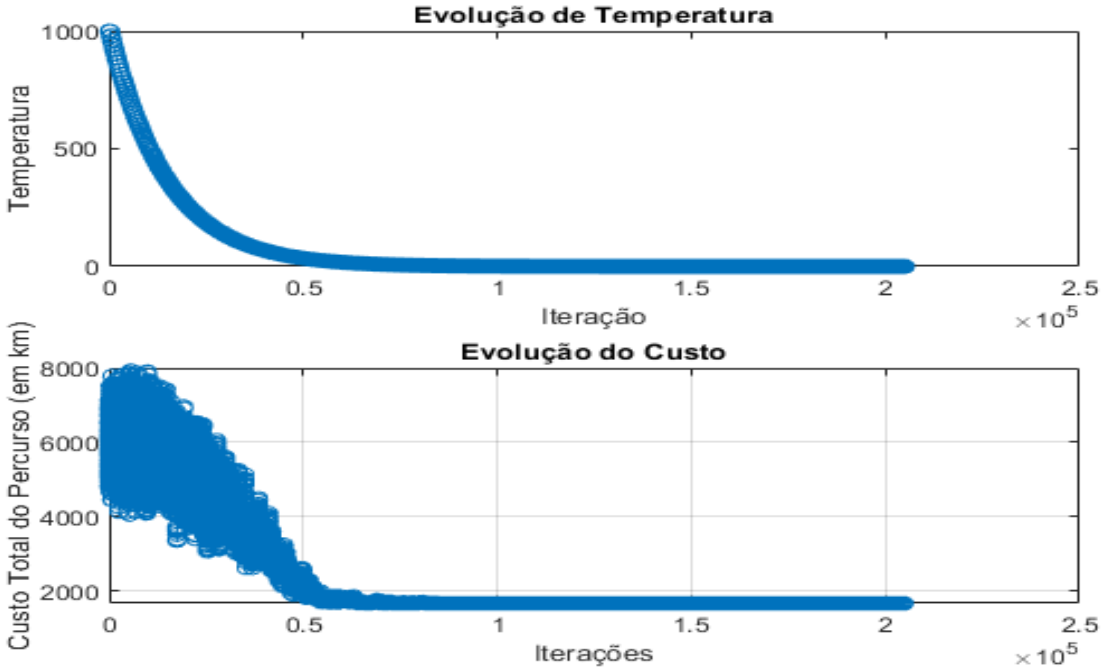

Figura 23-> 2º Teste 30 cidades: Gráficos de Estudo (Evolução Temperatura e Custo)

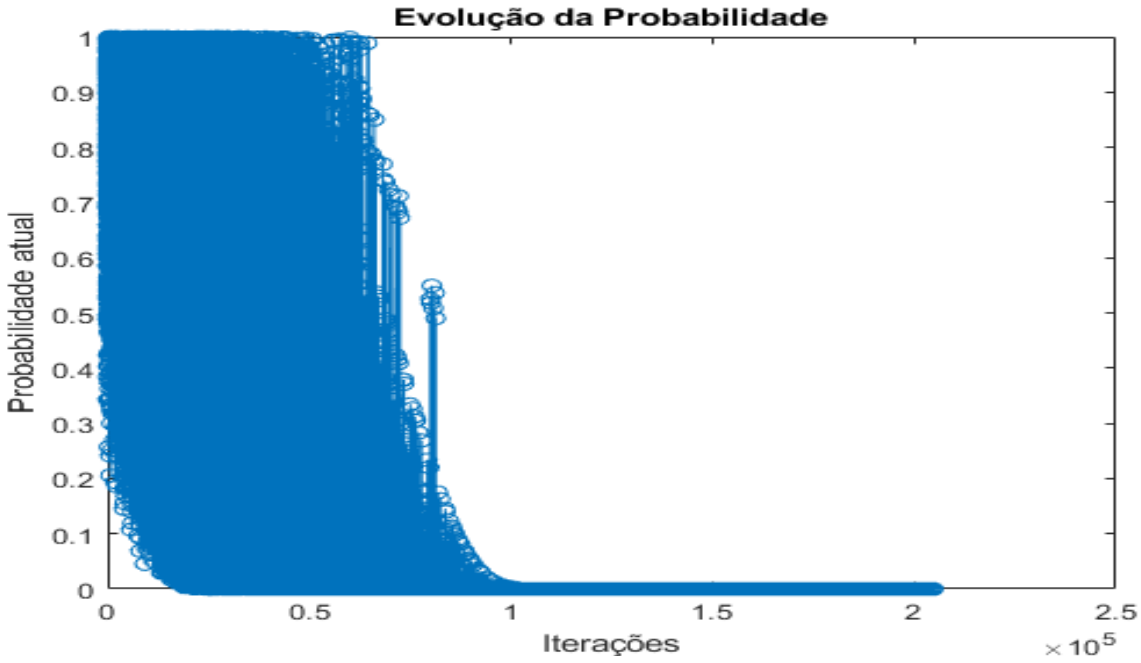

Figura 24-> 2º Teste 30 cidades: Gráficos de Estudo (Evolução da Probabilidade)

No segundo teste realizado com 30 cidades, os gráficos apresentados demonstram claramente a evolução do algoritmo de otimização ao longo das iterações. Inicialmente, o percurso gerado aleatoriamente apresenta um custo de 6532,77 km. Contudo, já nas primeiras iterações, observa-se uma redução expressiva do custo. Por exemplo, em torno da 17.000ª iteração, o custo foi reduzido para cerca de 3732,43 km.

À medida que o algoritmo avança, os ajustes tornam-se mais refinados. Na iteração número 51.127, percebe-se que o custo começa a estabilizar em torno de 1683,15 km, indicando que o processo de otimização está se aproximando de uma solução ótima. Essa estabilização sugere que o algoritmo atingiu um equilíbrio entre a exploração de novas soluções e a consolidação do melhor percurso. Na última iteração onde foi encontrado a melhor solução, de número 66.516, o percurso registado apresenta um custo de 1658,93 km, confirmando que o algoritmo continuou refinando a solução até alcançar um estado de estabilidade.

Os gráficos reforçam esses resultados. O gráfico de evolução da temperatura mostra uma redução gradual ao longo das iterações, refletindo o funcionamento do método de arrefecimento utilizado. Isso demonstra uma transição controlada do algoritmo, que inicialmente explora uma ampla gama de soluções e, posteriormente, concentra-se em ajustes locais mais detalhados. Já no gráfico de evolução do custo, nota-se uma redução inicial acentuada, seguida por uma fase de convergência, onde as oscilações diminuem significativamente até estabilizarem. Por fim, o gráfico de evolução da probabilidade mostra a diminuição na aceitação de soluções “piores”.

Esses resultados destacam a eficiência do algoritmo em encontrar um percurso altamente otimizado, reduzindo o custo inicial em mais de 70%. Ainda que a convergência para 1658,93 km seja bastante satisfatória, é válido considerar a possibilidade de ajustes nos parâmetros do algoritmo, como a taxa de arrefecimento ou o número máximo de iterações, para avaliar se é possível alcançar resultados ainda melhores.

4.7. Balanço de Resultados dos Algoritmos

Métrica	Hill Climbing	Hill Climbing Multiple Restart	Simulated Annealing
Melhor Valor Encontrado	(0.0655, 1.6334)	(0.0655, 1.6334)	(0.0655, 1.6334)
Média dos Valores	(0.7881, 0.4416)	(0.6356, 1.2199)	(0.1505,1.5773)
Pior Valor Encontrado	(0.8890, 0.0027)	(1.0752, 0.0932)	(1.4543,1.0199)
Número de Reinícios	N/A	21	N/A
Taxa de Convergência (%)	11%	43%	69%
Variação (%)	90.30%	69.42%	115.10%
Taxa de Aceitação	N/A	N/A	81.36%
Parâmetros Relevantes	-	-	Temperatura inicial= 90 α = 0.94

Figura 25-> Tabela de Comparação de Resultados

tabela. Foi iterado o código de cada algoritmo cem vezes e recolhidos os dados de modo que ao final das cem iterações fossem calculadas as médias para aqui as apresentar. Caso necessário, é possível executar novamente o código para confirmação de valores (ficheiros usados não estão anexados).

5. Conclusão

Este relatório analisou a implementação e o desempenho dos algoritmos Hill Climbing e Simulated Annealing em problemas de otimização, realizados no MATLAB. O Hill Climbing mostrou-se eficaz em encontrar máximos locais rapidamente, mas teve dificuldades em explorar espaços complexos. O Simulated Annealing, por outro lado, apresentou uma abordagem mais robusta, capaz de evitar máximos locais e alcançar soluções globais, especialmente em cenários desafiadores. A tabela criada para identificar os pontos fortes de cada algoritmo permitiu uma análise comparativa clara, ajudando a determinar a melhor abordagem para diferentes tipos de problemas de otimização. O estudo sugere que o Simulated Annealing é mais adequado para problemas com múltiplos máximos locais ou requisitos de uma exploração mais ampla do espaço de soluções.

Em suma, o grupo de trabalho encontra se satisfeito com o desenvolvimento deste, tendo cumprido os objetivos de aprendizagem que esta etapa obrigou.

6. Bibliografia

- Ariful, M. (2015). *Simulated Annealing and Hill Climbing in MCDC Test Case Generation*. ResearchGate. Disponível em: https://www.researchgate.net
- Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4ª ed.).
- Rao, S. S. (2009). *Engineering Optimization: Theory and Practice* (4ª ed.). Wiley.
- Knuth, D. (1997). *The Art of Computer Programming*. Addison-Wesley. Disponível em https://www-cs-faculty.stanford.edu/~knuth/taocp.html
- Oliveira, P. M. (2022). *Métodos de Pesquisa (Search) – Parte II*. Departamento de Engenharias, Universidade de Trás-os-Montes e Alto Douro (UTAD).

7. Anexos

7.1. Código MatLab “HillClimb.m”

```
close all;

clear all;


% Definição da função

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


% Geração dos pontos para o gráfico da função

x = linspace(0, 1.6, 200);

y = f(x);

plot(x, y, 'b');

hold on;


% Configurações iniciais

num_total = 1;

num_tent = 1;

f_evolucao = [];

x_evolucao = [];

f_tentativa = [];

j = 1;

i = 1;

delta = 0.02;

x_now = rand * 1.6;

x_old = x_now;


while i <= 300
```

```
    valor_rand = (rand - 0.5) * delta;
    x_now = x_old + 2 * valor_rand;

    % Verifica se x_now está dentro do intervalo [0, 1.6]
    if x_now < 0
        x_now = 0;
    elseif x_now > 1.6
        x_now = 1.6;
    end

    % Atualiza o valor de x se f(x_now) for maior
    if f(x_now) > f(x_old)
        plot(x_now, f(x_now), 'o'); % Plota o ponto
        x_old = x_now;
    end

    % Armazena a evolução de f(x) e de x
    f_evolucao(j) = f(x_old);
    x_evolucao(j) = x_old;
    i = i + 1;
    j = j + 1;
end

% Atualiza o número de tentativas e f_max
if (num_total == 1)
    num_total = num_total + 10;
    f_max = f(x_now);
```

```
    end
    if f(x_now) > f_max
        num_total = num_total + 10;
        f_max = f(x_now);
    end

    % Identificar o valor máximo da função e o x correspondente
    [f_max_value, idx_max] = max(f_evolucao); % Encontra o máximo em f_evolucao
    x_max = x_evolucao(idx_max); % Identifica o valor correspondente de x no vetor x_evolucao

    % Destacar o máximo no gráfico da função
    plot(x_max, f_max_value, '-o', 'MarkerSize', 10, ...
        'MarkerEdgeColor', 'red', 'MarkerFaceColor', 'yellow'); % Destaca o ponto no gráfico inicial
    title('Gráfico da Função');

    % Gráfico da evolução de f(x) e x
    figure;
    subplot(2, 1, 1);
    plot(1:(j - 1), f_evolucao); % Plota a evolução de f(x_now)
    hold on;
    [f_max_evolucao, idx_max_evolucao] = max(f_evolucao); % Máximo na evolução
    plot(idx_max_evolucao, f_max_evolucao, '-o', 'MarkerSize', 8, ...
        'MarkerEdgeColor', 'red', 'MarkerFaceColor', 'yellow');
    xlabel('Iteração');
    ylabel('f(x)');
    title('Evolução de f(x_{now}) a cada iteração');
```

```
hold off;


subplot(2, 1, 2);
plot(1:(j - 1), x_evolucao); % Plota a evolução de x
xlabel('Iteração');
ylabel('x');
title('Evolução de x a cada iteração');
```

7.2. Código MatLab “HillClimbMultipleRestart.m”

```
close all;
clear all;

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

% Geração dos pontos para o gráfico da função
x = linspace(0, 1.6, 200);
y = f(x);
plot(x, y, 'b');
hold on;

% Configurações iniciais
num_total = 1;
num_tent = 1;
f_evolucao = [];
x_evolucao = [];
f_tentativa = [];
```

```
j = 1;

% Loop principal
while num_tent <= 1
    i = 1;
    delta = 0.02;
    x_now = rand * 1.6;
    x_old = x_now;

    % Loop de atualização
    while i <= 300
        valor_rand = (rand - 0.5) * delta;
        x_now = x_old + 2 * valor_rand;

        % Verifica se x_now está dentro do intervalo [0, 1.6]
        if x_now < 0
            x_now = 0;
        elseif x_now > 1.6
            x_now = 1.6;
        end

        % Atualiza o valor de x se f(x_now) for maior
        if f(x_now) > f(x_old)
            plot(x_now, f(x_now), 'o'); % Plota o ponto
            x_old = x_now;
        end
```

```
            % Armazena a evolução de f(x) e de x
            f_evolucao(j) = f(x_old);
            x_evolucao(j) = x_old;
            i = i + 1;
            j = j + 1;
        end

        % Atualiza o número de tentativas e f_max
        if (num_total == 1)
            num_total = num_total + 10;
            f_max = f(x_now);
        end
        if f(x_now) > f_max
            num_total = num_total + 10;
            f_max = f(x_now);
        end
        f_tentativa(num_tent) = f(x_now);
        num_tent = num_tent + 1;
    end

    % Identificar o valor máximo da função e o x correspondente
    [f_max_value, idx_max] = max(f_evolucao); % Encontra o máximo em f_evolucao
    x_max = x_evolucao(idx_max); % Identifica o valor correspondente de x no vetor x_evolucao

    % Destacar o máximo no gráfico da função
    plot(x_max, f_max_value, '-o', 'MarkerSize', 10, ...
```

```
    'MarkerEdgeColor', 'red', 'MarkerFaceColor', 'yellow'); % Destaca o ponto no gráfico inicial
    title('Gráfico da Função');

    % Gráfico da evolução de f(x) e x
    figure;
    subplot(2, 1, 1);
    plot(1:(j - 1), f_evolucao); % Plota a evolução de f(x_now)
    hold on;
    [f_max_evolucao, idx_max_evolucao] = max(f_evolucao); % Máximo na evolução
    plot(idx_max_evolucao, f_max_evolucao, '-o', 'MarkerSize', 8, ...
        'MarkerEdgeColor', 'red', 'MarkerFaceColor', 'yellow');
    xlabel('Iteração');
    ylabel('f(x)');
    title('Evolução de f(x_{now}) a cada iteração');
    hold off;

    subplot(2, 1, 2);
    plot(1:(j - 1), x_evolucao); % Plota a evolução de x
    xlabel('Iteração');
    ylabel('x');
    title('Evolução de x a cada iteração');

    % Gráfico da evolução de f(x) máximo
    figure;
    plot(1:num_total, f_tentativa, '-o'); % Plota os valores
    hold on;
    [f_max_tent, idx_max_tent] = max(f_tentativa); % Máximo dos testes
```

```
plot(idx_max_tent, f_max_tent, '-o', 'MarkerSize', 10, ...
    'MarkerEdgeColor', 'red', 'MarkerFaceColor', 'yellow');
xlabel('Número Teste');
ylabel('f(x) max');
title('Evolução f(x) a cada teste');
legend('f(x)', 'Máximo Encontrado', 'Location', 'Best');
hold off;
```

7.3. Código MatLab “SimulatedAnnealing.m”

```
clear;
clc;
close all;
clear all;

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

% Intervalo e visualização inicial
x = linspace(0,1.6,100000);
y = f(x);
figure(1)
plot(x, y, 'b');
hold on;

% Parâmetros do SA
t = 1;
Tmax = 90;        % Temperatura maxima sugerida nos powerpoints do prof
```

```
T = Tmax;         % Temperatura inicial
alfa = 0.94;      % Fator de decaimento, parece melhor em 0.94 aqui
cicles = 300;     % Número de ciclos
Tit = 5;          % Iterações por temperatura
t_i = 1;          % Iterações do plot da temperatura
x_t = rand * 1.6; % Solução inicial aleatória
k = 0.8 * 1.6;    % Escalar do passo (tamanho máximo do primeiro passo)
x_max= x_t;       % maximo encontrado x
y_max = f(x_t);       % maximo encontrado y
f_evolucao = zeros(cicles, 1); % Evolução do x
fy_evolucao = zeros(cicles, 1); % Evolução do y
t_evolucao = zeros(cicles * Tit, 1); % Evolução da temperatura
p_evolucao = zeros(cicles * Tit, 1); % Evolução da probabilidade

while t <= cicles
   n = 1;
   while n <= Tit
      xi = k * ((T / Tmax) ^ 0.5); % tamanho do passo a dar
      x_new = x_t + (rand - 0.5) * xi; % x novo para testar
      x_new = max(0, min(1.6, x_new)); % confirmar intervalo
      dE = f(x_new) - f(x_t); % Calcula delta de energia
      p = exp(-abs(dE) / T); % Probabilidade de aceitação

      if dE >= 0 || rand < p % Critérios de movimentação
         x_t = x_new; % se aceite muda

      if f(x_t) > y_max
```

```
            y_max = f(x_t);
            x_max = x_t;
        end

        end

        f_evolucao(t) = x_t; % guarda x atual num array
        fy_evolucao(t) = f(x_t);
        plot(x_t, f(x_t), 'ro'); % Visualiza ponto atual
        n = n + 1;

        t_evolucao(t_i) = T;
        p_evolucao(t_i) = p;
        t_i = t_i + 1;
    end

    T = alfa * T; % Atualiza temperatura e armazena evolução
    t = t + 1;
end

plot(x_t, f(x_t), 'go'); % último ponto
plot(x_max, y_max, 'ks', 'LineWidth', 2); % ponto máximo

% Gráfico da evolução da função
figure(2);
plot(1:cicles, f(f_evolucao), '-o');
xlabel('Iteração');
```

```
ylabel('f(x_{now})');
title('Evolução de f(x_{now})');
grid on;

% Gráfico da evolução da temperatura
figure(3);
plot(1:length(t_evolucao), t_evolucao, '-o');
xlabel('Iteração');
ylabel('Temperatura');
title('Evolução de Temperatura');

% Gráfico da evolução da probabilidade
figure(4);
plot(1:length(t_evolucao), p_evolucao, '-o');
xlabel('Iteração');
ylabel('Probabilidade');
title('Evolução de Probabilidade');
```

7.4. Código MatLab “Caix_Viajante/Main.m”

```
clc;
clear all;
close all;
fprintf("1->14 cidades\n2->20 cidades\n3->30 cidades\n")
while true
    opcao = input('Por favor, introduza uma opção (1 | 2 | 3): ');
    if isnumeric(opcao) && ismember(opcao, [1, 2, 3])
        break;
```

```
    end
end
if opcao == 1
    % Coordenadas das cidades
    coordenadas_cities = [
        1 41.49 -6.45;  % Bragança
        2 41.18 -7.45;  % Vila Real
        3 41.44 -7.28;  % Chaves
        4 41.27 -8.18;  % Guimarães
        5 41.33 -8.26;  % Braga
        6 41.42 -8.50;  % Viana do Castelo
        7 42.02 -8.38;  % Valença
        8 41.32 -8.37;  % Barcelos
        9 41.11 -8.36;  % Porto
        10 40.38 -8.39; % Aveiro
        11 40.34 -8.27; % Águeda
        12 40.39 -7.55; % Viseu
        13 41.06 -7.49; % Lamego
        14 41.10 -7.47; % Peso Régua
    ];
    num_cidades = 14;

    % Nomes das cidades
    nomes = { 'Bragança', 'Vila Real', 'Chaves', 'Guimarães', 'Braga', ...
            'Viana do Castelo', 'Valença do Minho', 'Barcelos', 'Porto', ...
            'Aveiro', 'Águeda', 'Viseu', 'Lamego', 'Peso da Régua' };
    alpha = 0.96;         % Fator de redução da temperatura
```

```
    iteracoes = 50;       % Iterações por temperatura
    T_inicial = 90;       % Temperatura inicial
end
if opcao==2
    % Coordenadas das cidades
    coordenadas_cities = [
        1 41.49 -6.45;  % Bragança
        2 41.18 -7.45;  % Vila Real
        3 41.44 -7.28;  % Chaves
        4 41.42 -8.50;  % Viana do Castelo
        5 41.33 -8.26;  % Braga
        6 40.38 -8.39;  % Aveiro
        7 41.11 -8.36;  % Porto
        8 40.39 -7.55;  % Viseu
        9 41.06 -7.49;  % Lamego
        10 41.27 -8.18; % Guimarães
        11 40.12 -8.25; % Coimbra
        12 37.01 -7.56; % Faro
        13 38.34 -7.54; % Évora
        14 38.43 -9.10; % Lisboa
        15 39.17 -7.26; % Portalegre
        16 37.07 -7.39; % Tavira
        17 37.00 -8.56; % Sagres
        18 38.32 -8.54; % Setúbal
        19 40.32 -7.16; % Guarda
        20 39.14 -8.41; % Santarém
    ];
```

```
    num_cidades = 20;

    % Nomes das cidades
    nomes = { 'Bragança', 'Vila Real', 'Chaves', 'Viana do Castelo', 'Braga', ...
            'Aveiro', 'Porto', 'Viseu', 'Lamego', 'Guimarães', ...
            'Coimbra', 'Faro', 'Évora', 'Lisboa', 'Portalegre', ...
            'Tavira', 'Sagres', 'Setúbal', 'Guarda', 'Santarém' };
    alpha = 0.98;         % Fator de redução da temperatura
    iteracoes = 100;      % Iterações por temperatura
    T_inicial = 90;       % Temperatura inicial
end
if opcao==3
    % Coordenadas das cidades
    coordenadas_cities = [
        1 41.49 -6.45;  % Bragança
        2 41.18 -7.45;  % Vila Real
        3 41.44 -7.28;  % Chaves
        4 41.42 -8.50;  % Viana do Castelo
        5 41.33 -8.26;  % Braga
        6 40.38 -8.39;  % Aveiro
        7 41.11 -8.36;  % Porto
        8 40.39 -7.55;  % Viseu
        9 41.06 -7.49;  % Lamego
        10 41.27 -8.18; % Guimarães
        11 40.12 -8.25; % Coimbra
        12 37.01 -7.56; % Faro
        13 38.34 -7.54; % Évora
```

```
    14 38.43 -9.10; % Lisboa
    15 39.17 -7.26; % Portalegre
    16 37.07 -7.39; % Tavira
    17 37.00 -8.56; % Sagres
    18 38.32 -8.54; % Setúbal
    19 40.32 -7.16; % Guarda
    20 39.14 -8.41; % Santarém
    21 38.01 -7.52; % Beja
    22 37.57 -8.52; % Sines
    23 40.17 -7.30; % Covilhã
    24 39.36 -8.25; % Tomar
    25 40.34 -8.27; % Águeda
    26 39.45 -8.48; % Leiria
    27 39.49 -7.30; % Castelo Branco
    28 38.53 -7.10; % Elvas
    29 41.30 -6.16; % Miranda do Douro
    30 38.48 -9.23; % Sintra
];
num_cidades = 30;

% Nomes das cidades
nomes = { 'Bragança', 'Vila Real', 'Chaves', 'Viana do Castelo', 'Braga', ...
          'Aveiro', 'Porto', 'Viseu', 'Lamego', 'Guimarães', ...
          'Coimbra', 'Faro', 'Évora', 'Lisboa', 'Portalegre', ...
          'Tavira', 'Sagres', 'Setúbal', 'Guarda', 'Santarém', ...
          'Beja', 'Sines', 'Covilhã', 'Tomar', 'Águeda', ...
          'Leiria', 'Castelo Branco', 'Elvas', 'Miranda do Douro', 'Sintra' };
```

```
    alpha = 0.98;         % Fator de redução da temperatura
    iteracoes = 300;      % Iterações por temperatura
    T_inicial = 1000;     % Temperatura inicial
end
% Coordenadas em matriz
cities = [coordenadas_cities(:, 2)'; coordenadas_cities(:, 3)'];

% Distância Haversine
R_Terra = 6376; %valor do raio da terra para calcular custo em km
distancia = @(c1, c2) 2 * R_Terra * ...
    asin(sqrt(sin((deg2rad(c2(1)) - deg2rad(c1(1))) / 2)^2 + ...
            cos(deg2rad(c1(1))) * cos(deg2rad(c2(1))) * ...
            sin((deg2rad(c2(2)) - deg2rad(c1(2))) / 2)^2));

% Função J (soma total das distâncias no percurso)
calculaCusto = @(percurso) sum(arrayfun(@(i) ...
    distancia(cities(:, percurso(i)), cities(:, percurso(mod(i, num_cidades) + 1))), ...
    1:num_cidades));

% Parâmetros do Simulated Annealing

T_final = 1e-3;       % Temperatura final

historico_temperatura = [];  % Vetor para guardar a temperatura ao longo das iterações
historico_custos = [];
historico_probs=[];
it=0;
```

```
% Inicialização do percurso fixando a cidade inicial (cidade 1 como exemplo)
percursoAtual = randperm(num_cidades);
custoAtual = calculaCusto(percursoAtual);
melhorpercurso = percursoAtual;
melhorcusto = custoAtual;

% Iteradores
Tit = T_inicial;
it=0;
% Loop principal
while Tit > T_final
    i = 0;
    while i < iteracoes
        % Geração de vizinho mantendo a cidade inicial fixa
        vizinho = percursoAtual;
        idx = randperm(num_cidades-1, 2) + 1; % Evita a cidade inicial (fixa em 1)
        vizinho(idx) = vizinho(flip(idx));

        custoVizinho = calculaCusto(vizinho);
        delta = custoVizinho - custoAtual;

        % Critério de aceitação
        if delta < 0 || rand < exp(-abs(delta) / Tit)
            percursoAtual = vizinho;
            custoAtual = custoVizinho;
        end
```

```
            % Atualização do melhor percurso
            if custoAtual < melhorcusto
                melhorpercurso = percursoAtual;
                melhorcusto = custoAtual;
                figure(1); clf;
                plot(cities(2, percursoAtual), cities(1, percursoAtual), '-o', ...
                    'LineWidth', 1, 'Color', [0.2 0.2 0.8], 'MarkerSize', 6);
                hold on;
                plot([cities(2, percursoAtual(end)), cities(2, percursoAtual(1))], ...
                     [cities(1, percursoAtual(end)), cities(1, percursoAtual(1))], '-', 'Color', [0.2
0.2 0.8]);

                for j = 1:num_cidades
                    text(cities(2, percursoAtual(j)), cities(1, percursoAtual(j)), ...
                        nomes{percursoAtual(j)}, 'FontSize', 8, ...
                        'HorizontalAlignment', 'center', 'VerticalAlignment', 'bottom');
                end

                scatter(cities(2, percursoAtual(1)), cities(1, percursoAtual(1)), 100, 'r', 'filled');
                title(sprintf('Iteração %d - Custo: %.2f km', it, custoAtual));
                xlabel('Longitude');
                ylabel('Latitude');
                grid on; axis equal;
                drawnow;
            end

            % Armazenamento de histórico
            historico_probs = [historico_probs, exp(-abs(delta) / Tit)];
```

```
        historico_temperatura = [historico_temperatura, Tit];
        historico_custos = [historico_custos, custoAtual];
        i = i + 1;
        it=it+1;
    end

    Tit = Tit * alpha;
end
close all;



% Reestruturar as coordenadas
cities = [coordenadas_cities(:, 2), coordenadas_cities(:, 3)]; % [latitude, longitude]

% Plotar o percurso corretamente
figure;
hold on;

plot(cities(melhorpercurso, 2), cities(melhorpercurso, 1), '-o', ...
    'LineWidth', 1, 'Color', [0.2 0.2 0.8], 'MarkerSize', 6);

% Plot para conectar a ultima cidade à primeira (ciclo)
plot([cities(melhorpercurso(end), 2), cities(melhorpercurso(1), 2)], [cities(melhorpercurso(end), 1), cities(melhorpercurso(1), 1)], '-', 'LineWidth', 1, 'Color', [0.2 0.2 0.8]);

% Ciclo for para adicionar nomes das cidades apartir da lista "melhor percurso"
```

```
for i = 1:num_cidades
    text(cities(melhorpercurso(i), 2), cities(melhorpercurso(i), 1), ...
        nomes{melhorpercurso(i)}, ...
        'FontSize', 8, 'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'bottom', 'Color', 'k');
end

% Cidade de Partida no melhor percurso
scatter(cities(melhorpercurso(1), 2), cities(melhorpercurso(1), 1), 100, ...
       'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'LineWidth', 1);
text(cities(melhorpercurso(1), 2), cities(melhorpercurso(1), 1), ...
    'Início', 'FontSize', 10, 'FontWeight', 'bold', 'Color', 'r', ...
    'HorizontalAlignment', 'center', 'VerticalAlignment', 'top');

% Configuração Chat
title('Melhor Percurso Encontrado', 'FontSize', 12, 'FontWeight', 'bold');
xlabel('Longitude', 'FontSize', 10);
ylabel('Latitude', 'FontSize', 10);
grid on;
axis equal;
xlim([min(cities(:, 2)) - 0.5, max(cities(:, 2)) + 0.5]);
ylim([min(cities(:, 1)) - 0.5, max(cities(:, 1)) + 0.5]);
set(gca, 'FontSize', 10);
hold off;
% Até aqui

% Gráfico da evolução da temperatura e do custo
```

```
figure;

% Subplot 1: Evolução da temperatura
subplot(2, 1, 1);
plot(1:length(historico_temperatura), historico_temperatura, '-o');
xlabel('Iteração');
ylabel('Temperatura');
title('Evolução de Temperatura');

% Subplot 2: Evolução do custo
subplot(2, 1, 2);
plot(1:length(historico_custos), historico_custos, '-o');
xlabel('Iterações');
ylabel('Custo Total do Percurso (em km)');
title('Evolução do Custo');
grid on;

% Subplot 3: Evolução da probabilidade
figure;
plot(1:length(historico_probs), historico_probs, '-o');
xlabel('Iterações');
ylabel('Probabilidade atual');
title('Evolução da Probabilidade');


% Ajuste geral
set(gca, 'FontSize', 10);
```