

**Análise Algorítmica Aplicada ao TSP Estratégias e Resultados**

**António Silva - 8220207**

**Tiago Pacheco - 8220208**

**Guilherme Barreiro – 8220849**

**Carlos Barbosa - 8210417**

**Análise Algorítmica e Otimização**

# Índice

[Índice ii](#_Toc198676031)

[Índice de Figuras iv](#_Toc198676032)

[Índice de Tabelas v](#_Toc198676033)

[Lista de Siglas e Acrónimos vi](#_Toc198676034)

[1. Introdução 1](#_Toc198676035)

[1.1 Contextualização 1](#_Toc198676036)

[1.2 Apresentação do Caso de Estudo 1](#_Toc198676037)

[1.3 Motivação e Objetivos 2](#_Toc198676038)

[1.4 Estrutura do Relatório 2](#_Toc198676039)

[2. Pesquisa bibliográfica sobre o Problema do Caixeiro Viajante 3](#_Toc198676040)

[3. Implementação de algoritmos para a resolução do Problema do Caixeiro Viajante 4](#_Toc198676041)

[3.1 Algoritmo Vizinho Mais Próximo (Nearest Neighbor) 4](#_Toc198676042)

[3.2 Inserção com Menor Custo (Cheapest Insertion): 5](#_Toc198676043)

[3.3 Inserção Mais Longínqua (Farthest Insertion) 7](#_Toc198676044)

[3.4 Aproximação por Crescimento de Árvore (Minimum Spanning Tree Heuristic – MST) 8](#_Toc198676045)

[3.5 Heurística do Caminho Aleatório (Random Path Construction) 9](#_Toc198676046)

[4. Implementação de algoritmos para o melhoramento da resolução do Problema do Caixeiro Viajante 9](#_Toc198676047)

[4.1 2-Opt 9](#_Toc198676048)

[4.2 3-Opt 11](#_Toc198676049)

[4.3 3-OptBestOption 12](#_Toc198676050)

[4.4 Or-Opt 14](#_Toc198676051)

[4.5 K-Opt 14](#_Toc198676052)

[4.6 Movimentação de Subsequências (Lin-Kernighan Heuristic) 16](#_Toc198676053)

[5. Análise do desempenho dos algoritmos implementados 17](#_Toc198676054)

[5.1 Qualidade das Soluções 23](#_Toc198676055)

[5.2 Tempo Computacional 24](#_Toc198676056)

[5.3 Conclusões Parciais e Recomendações 24](#_Toc198676057)

[6. Conclusões e Trabalho Futuro 26](#_Toc198676058)

[Referências Bibliográficas 27](#_Toc198676059)

[Referências WWW 28](#_Toc198676060)

# Índice de Figuras

[Figura 1: Nearest Neighbor 5](#_Toc198676061)

[Figura 2: Cheapest Insertion 6](#_Toc198676062)

[Figura 3: Farthest Insertion 7](#_Toc198676063)

[Figura 4: Minimum Spanning Tree Heuristic – MST 8](#_Toc198676064)

[Figura 5: Random Path Construction 9](#_Toc198676065)

[Figura 6: 2-Opt 10](#_Toc198676066)

[Figura 7: 3-Opt 11](#_Toc198676067)

[Figura 8:Algoritmo 3Opt Best Option 13](#_Toc198676068)

[Figura 9: Or-Opt 14](#_Toc198676069)

[Figura 10: K-Opt 15](#_Toc198676070)

[Figura 11: Lin-Kernighan Heuristic) 16](#_Toc198676071)

# Índice de Tabelas

[Tabela 1: Siglas e o seu significado vi](#_Toc198676072)

[Tabela 2: Resultados resumidos 23](#_Toc198676073)

# Lista de Siglas e Acrónimos

|  |  |
| --- | --- |
| Sigla | Significado |
| BD | Base de Dados |
| SGBD | Sistema de Gestão de Base de Dados |
| TSP | *Travelling Salesman Problem* (Problema do Caixeiro Viajante) |
| MST | *Minimum Spanning Tree* (Árvore Geradora Mínima) |
| NN | *Nearest Neighbor* (Vizinho Mais Próximo) |
| CI | *Cheapest Insertion* (Inserção com Menor Custo) |
| FI | *Farthest Insertion* (Inserção Mais Longínqua) |
| LK | *Lin-Kernighan* (Heurística de Lin-Kernighan) |
| ACO | *Ant Colony Optimization* (Colónia de Formigas) |
| GA | *Genetic Algorithm* (Algoritmo Genético) |
| SA | *Simulated Annealing* (Recozimento Simulado) |
| SO | Solução Ótima |
| SE | Solução Encontrada |
| CPU | *Central Processing Unit* (Unidade Central de Processamento) |
| TSPLIB | Biblioteca de Instâncias para o TSP |
| OptK | Heurística genérica de k-Opt |
| 2-Opt | Heurística de melhoria com 2 trocas |
| 3-Opt | Heurística de melhoria com 3 trocas |
| Or-Opt | Heurística de movimentação de subsequências |

Tabela 1: Siglas e o seu significado

# 1. Introdução

O presente documento tem como objetivo apresentar o desenvolvimento de um trabalho prático relacionado com o Problema do Caixeiro Viajante (TSP), focando-se na análise de diferentes algoritmos de resolução. Serão exploradas várias abordagens — desde heurísticas construtivas a métodos de melhoria local —, avaliando o seu desempenho com base em métricas como qualidade da solução e tempo de execução.

O tipo de letra utilizado neste relatório é Arial, com exceção de excertos de código, que serão apresentados em Courier New. Os termos estrangeiros são destacados em itálico ou entre aspas, de acordo com as recomendações de boas práticas académicas.

## 1.1 Contextualização

A resolução de problemas de otimização combinatória é um dos pilares da ciência da computação. O TSP, em particular, é um problema clássico amplamente estudado devido à sua simplicidade de enunciação e complexidade de resolução. Este problema é classificado como NP-difícil e tem aplicações em áreas como logística, planeamento de circuitos eletrónicos e bioinformática. A sua importância prática e teórica justifica a escolha deste tema para análise algorítmica.

## 1.2 Apresentação do Caso de Estudo

O caso de estudo selecionado para este trabalho é o *Problema do Caixeiro Viajante (TSP)*, amplamente reconhecido pela sua relevância teórica e prática em áreas como logística, planeamento de rotas, otimização de processos, e muitas outras aplicações do quotidiano. A complexidade inerente ao TSP, classificado como *NP-difícil*, o que o torna um desafio aliciante para a aplicação de algoritmos e estruturas de dados eficientes.

O desenvolvimento deste trabalho surgiu da necessidade de explorar, comparar e avaliar diferentes técnicas de resolução do TSP, desde métodos exatos até heurísticas e meta-heurísticas. Assim, pretende-se não só compreender os fundamentos matemáticos e computacionais que sustentam este problema, mas também implementar algoritmos que sejam capazes de encontrar soluções de boa qualidade em tempos de execução razoáveis. Deste modo podemos não só implementar e desenvolver algoritmos capazes de solucionar o problema, como também analisar essas mesmas resoluções e concluir a sua eficiência.

Deste modo, os objetivos principais deste estudo podem ser sintetizados em três pontos fundamentais:

* **Explorar a teoria e os algoritmos clássicos** do TSP (força bruta, *branch and bound*, programação dinâmica, entre outros), analisando as suas vantagens e limitações.
* **Implementar heurísticas e meta-heurísticas** (Lin-Kernighan, Colónia de Formigas, Algoritmos Genéticos, etc.) e avaliar o seu desempenho com base em critérios como tempo de execução e qualidade das soluções obtidas.
* **Comparar resultados e discutir melhorias** possíveis, seja pela otimização dos algoritmos escolhidos ou pela adoção de estruturas de dados avançadas que permitam acelerar e refinar a busca de soluções.

Posto isto, este caso de estudo oferece uma excelente oportunidade para aprofundar conhecimentos em otimização combinatória e implementação de algoritmos, enquanto se aborda um problema com aplicações práticas em diversos setores industriais e científicos.

## 1.3 Motivação e Objetivos

A motivação principal deste trabalho prende-se com a oportunidade de aplicar e consolidar conhecimentos adquiridos nas áreas de algoritmia, estruturas de dados e otimização combinatória. O Problema do Caixeiro Viajante (TSP), sendo um problema clássico e de elevada complexidade, oferece um contexto ideal para experimentar diferentes abordagens algorítmicas e avaliar criticamente a sua eficácia.

Entre os objetivos específicos, destacam-se:

* A implementação e análise comparativa de algoritmos heurísticos construtivos, que produzem rapidamente soluções iniciais.
* A aplicação de algoritmos de melhoria local e meta-heurísticas para refinar essas soluções e reduzir o custo total dos percursos.
* A avaliação empírica do desempenho dos algoritmos, com base em métricas como desvio percentual em relação à solução ótima e tempo de execução.
* A criação de um sistema modular que permita reutilizar código e aplicar as soluções desenvolvidas a múltiplas instâncias do problema.

## 1.4 Estrutura do Relatório

Este relatório está organizado em seis secções principais. Na Secção 1, é feita uma introdução ao trabalho, incluindo a contextualização, os objetivos e a motivação por detrás da sua realização. A Secção 2 apresenta a pesquisa bibliográfica efetuada, focando-se na origem, complexidade e diferentes abordagens para resolução do TSP. Na Secção 3, são descritas as heurísticas construtivas implementadas, seguidas pela Secção 4, que aborda as heurísticas de melhoria local utilizadas para refinar as soluções iniciais. A Secção 5 dedica-se à análise detalhada dos resultados obtidos, comparando a eficácia e eficiência dos diferentes métodos. Por fim, a Secção 6 oferece uma reflexão crítica sobre o trabalho desenvolvido e propõe direções para trabalho futuro.

# 2. Pesquisa bibliográfica sobre o Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (Travelling Salesman Problem - TSP) é um dos desafios clássicos da otimização combinatória e consiste em **encontrar o caminho mais curto** que um caixeiro-viajante deve percorrer para **visitar um conjunto de cidades exatamente uma vez e regressar à cidade de origem**.

Este problema é NP-difícil, o que significa que **não existe um algoritmo eficiente conhecido que resolva todas as instâncias em tempo polinomial**, uma vez que a complexidade cresce exponencialmente, tornando inviável a resolução de grandes instâncias com métodos exatos.

Para resolver o TSP, existem abordagens **exatas e heurísticas/meta-heurísticas**. Os **métodos exatos, como força bruta, programação dinâmica (Held-Karp), branch and bound e programação linear**, garantem a solução ótima, mas tornam-se inviáveis à medida que o número de cidades aumenta. Por esse motivo, são amplamente utilizados algoritmos heurísticos e meta-heurísticos, que encontram boas soluções em tempo razoável.

Entre as **heurísticas** mais simples destacam-se o **vizinho mais próximo, a inserção mais barata e o algoritmo de Christofides**, que garante um erro máximo de 50% para instâncias simétricas. No entanto, para soluções mais refinadas, recorrem-se a métodos mais sofisticados como Lin-Kernighan (LK), um dos algoritmos mais eficazes baseados em busca local, e o Stem-and-Cycle (S&C), uma abordagem mais recente que pode superar o LK em certos casos. Além disso, meta-heurísticas como Busca Tabu, Simulated Annealing, **Algoritmos Genéticos e Colónia de Formigas (ACO) são amplamente utilizadas para lidar com instâncias mais complexas.**

O desempenho destes algoritmos depende fortemente das estruturas de dados utilizadas. **2-Level Trees e k-Level Satellite Trees melhoram a eficiência da manipulação das soluções, enquanto técnicas como listas de vizinhança restrita (k-nearest neighbors, don’t-look-bits) reduzem o espaço de busca e aceleram a execução sem comprometer a qualidade das soluções.** Estas abordagens permitem que heurísticas modernas resolvam instâncias muito maiores do TSP do que era possível há algumas décadas.

O TSP tem muitas aplicações práticas, incluindo otimização de rotas na logística e transportes, planeamento de circuitos eletrónicos, robótica, bioinformática, entre outras áreas que exigem soluções eficientes para problemas de roteamento. A combinação de algoritmos heurísticos avançados e estruturas de dados otimizadas continua a ser fundamental para resolver instâncias reais deste problema de forma eficiente.

# 3. Implementação de algoritmos para a resolução do Problema do Caixeiro Viajante

O carregamento de instâncias TSPLIB e o cálculo dos custos de percurso são feitos através de três componentes na classe **Utils**:

* **Representação das cidades**

A classe estática aninhada **City** guarda, para cada cidade, um identificador inteiro (id) e as coordenadas X e Y (números reais).

O construtor recebe (id, x, y) e existe o método distanceTo(City other) que retorna a distância Euclidiana entre duas cidades.

* **Leitura de ficheiro .tsp**

O método readTSPFile(String fileName) abre o ficheiro e procura a linha NODE\_COORD\_SECTION.

A seguir, até encontrar o marcador EOF, cada linha é dividida em três campos:

o primeiro convertido em inteiro (identificador),

os dois seguintes em real (coordenadas).

Para cada linha válida, cria-se um objeto City que se adiciona a uma lista devolvida ao final.

* **Cálculo do custo de um tour**

O método calculatePathCost(List<City> tour) percorre a sequência de cidades (incluindo o regresso à inicial) somando, para cada par consecutivo, a distância gerada por distanceTo.

O valor resultante é o comprimento total do percurso, usado para comparar soluções e medir desvios face ao ótimo conhecido.

## 3.1 Algoritmo Vizinho Mais Próximo (Nearest Neighbor)

Parte de uma cidade arbitrária e, em cada etapa, seleciona a cidade mais próxima ainda não visitada. É superrápido e fácil de implementar, mas pode ficar preso em soluções desfavoráveis se o ponto de partida não for adequado.

A screenshot of a computer program

AI-generated content may be incorrect.

Figura 1: Nearest Neighbor

## 3.2 Inserção com Menor Custo (Cheapest Insertion):

Inicia com um sub-percurso (geralmente duas cidades) e, a cada iteração, insere a cidade que, quando incluída entre duas já existentes, gera o menor aumento no custo total do percurso.

A screenshot of a computer program

AI-generated content may be incorrect.

Figura : Cheapest Insertion

## 3.3 Inserção Mais Longínqua (Farthest Insertion)

Parte de uma cidade e da cidade mais distante dela. Em seguida, insere iterativamente a cidade que se encontra mais afastada do percurso atual, tentando “forçar” a inclusão de pontos distantes para evitar deslocações longas no final.

A screenshot of a computer program

AI-generated content may be incorrect.

Figura 3: Farthest Insertion

## 3.4 Aproximação por Crescimento de Árvore (Minimum Spanning Tree Heuristic – MST)

Constrói uma árvore geradora mínima com todas as cidades e depois converte a árvore em um percurso fechado. É um método que garante soluções rápidas e razoáveis, mas não necessariamente ótimas.

A screenshot of a computer program

AI-generated content may be incorrect.

Figura : Minimum Spanning Tree Heuristic – MST

## 3.5 Heurística do Caminho Aleatório (Random Path Construction)

Forma um percurso inicial simplesmente escolhendo cidades aleatoriamente até cobrir todas as cidades. Serve como baseline para comparação com outras heurísticas.

A screen shot of a computer code

AI-generated content may be incorrect.

Figura 5: Random Path Construction

# 4. Implementação de algoritmos para o melhoramento da resolução do Problema do Caixeiro Viajante

## 4.1 2-Opt

Remove dois segmentos do percurso e reconecta-os de forma a eliminar cruzamentos desnecessários. É simples, mas geralmente proporciona melhorias significativas ao eliminar trajetos redundantes ou “ziguezagues”.

A screen shot of a computer

AI-generated content may be incorrect.

Figura 6: 2-Opt

## 4.2 3-Opt

É uma extensão do 2-Opt onde três segmentos são trocados. Embora seja mais exigente a nível computacional, explora uma vizinhança maior e pode escapar de ótimos locais onde o 2-Opt se limita.

A screenshot of a computer program

AI-generated content may be incorrect.

Figura 7: 3-Opt

## 4.3 3-OptBestOption

É uma extensão do 2-Opt onde três segmentos são trocados. Embora seja mais exigente a nível computacional, explora uma vizinhança maior e pode escapar de ótimos locais onde o 2-Opt se limita. Única diferença é que esta versão usa a *best option* e não a f*irst option*.

A screen shot of a computer

AI-generated content may be incorrect.

Figura :Algoritmo 3Opt Best Option

## 4.4 Or-Opt

Realiza movimentações de subsequências de cidades (normalmente de tamanho 1, 2 ou 3) para outras posições no percurso, visando reduzir o custo total. Este método é um bom complemento aos movimentos 2-Opt/3-Opt.

A screenshot of a computer screen

AI-generated content may be incorrect.

Figura 9: Or-Opt

## 4.5 K-Opt

Realiza movimentações de subsequências de cidades (normalmente de tamanho 1, 2 ou 3) para outras posições no percurso, visando reduzir o custo total. Este método é um bom complemento aos movimentos 2-Opt/3-Opt.

A screen shot of a computer

AI-generated content may be incorrect.

Figura 10: K-Opt

## 4.6 Movimentação de Subsequências (Lin-Kernighan Heuristic)

Uma técnica sofisticada que expande o conceito de k-Opt, alternando entre diferentes segmentos de forma adaptativa para encontrar uma boa solução.

A screenshot of a computer program

AI-generated content may be incorrect.

Figura 11: Lin-Kernighan Heuristic)

# 5. Análise do desempenho dos algoritmos implementados

Nesta secção discutimos a qualidade das soluções obtidas e o tempo de execução dos diferentes métodos de pós-processamento aplicados à solução inicial gerada pela heurística NearestNeighbor, com base nos resultados resumidos da tabela seguinte:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Instancia | Solução Otima (SO) | Solução inicial | Solução encontrada (SE) | % de desvio | Tempo computacional | Heuristica Inicial | Algoritmo de processamento |
| a280 | 2579 | 3324.59 | 2879.81 | 11.6638232 | 14 | NearestNeighbor | LinKernighanHeuristic |
| a280 | 2579 | 3324.59 | 3324.59 | 28.9100427 | 1 | NearestNeighbor | OptK |
| a280 | 2579 | 3324.59 | 2880.54 | 11.6921287 | 3329 | NearestNeighbor | OptOr |
| a280 | 2579 | 3324.59 | 3130.3 | 21.3765025 | 10061 | NearestNeighbor | Opt3 |
| bier127 | 118282 | 135098.7 | 124973.92 | 5.65759794 | 0 | NearestNeighbor | LinKernighanHeuristic |
| bier127 | 118282 | 135098.7 | 135098.7 | 14.2174634 | 0 | NearestNeighbor | OptK |
| bier127 | 118282 | 135098.7 | 121275.43 | 2.530757 | 203 | NearestNeighbor | OptOr |
| bier127 | 118282 | 135098.7 | 129692.59 | 9.64693698 | 605 | NearestNeighbor | Opt3 |
| ch130 | 6110 | 7575.29 | 6741.95 | 10.3428805 | 0 | NearestNeighbor | LinKernighanHeuristic |
| ch130 | 6110 | 7575.29 | 7575.29 | 23.9818331 | 0 | NearestNeighbor | OptK |
| ch130 | 6110 | 7575.29 | 6900.38 | 12.9358429 | 177 | NearestNeighbor | OptOr |
| ch130 | 6110 | 7575.29 | 6741.39 | 10.3337152 | 1195 | NearestNeighbor | Opt3 |
| ch150 | 6528 | 8194.61 | 6665.32 | 2.10355392 | 0 | NearestNeighbor | LinKernighanHeuristic |
| ch150 | 6528 | 8194.61 | 8194.61 | 25.5301777 | 0 | NearestNeighbor | OptK |
| ch150 | 6528 | 8194.61 | 7094.48 | 8.67769608 | 252 | NearestNeighbor | OptOr |
| ch150 | 6528 | 8194.61 | 7382.81 | 13.0945159 | 693 | NearestNeighbor | Opt3 |
| d198 | 15780 | 18585.12 | 16363.13 | 3.69537389 | 0 | NearestNeighbor | LinKernighanHeuristic |
| d198 | 15780 | 18585.12 | 18585.12 | 17.7764259 | 0 | NearestNeighbor | OptK |
| d198 | 15780 | 18585.12 | 17465.07 | 10.6785171 | 831 | NearestNeighbor | OptOr |
| d198 | 15780 | 18585.12 | 17258.92 | 9.3721166 | 3332 | NearestNeighbor | Opt3 |
| eil101 | 629 | 892.22 | 707.74 | 12.518283 | 0 | NearestNeighbor | LinKernighanHeuristic |
| eil101 | 629 | 892.22 | 892.22 | 41.8473768 | 0 | NearestNeighbor | OptK |
| eil101 | 629 | 892.22 | 736.59 | 17.1049285 | 178 | NearestNeighbor | OptOr |
| eil101 | 629 | 892.22 | 745.06 | 18.4515103 | 203 | NearestNeighbor | Opt3 |
| gil262 | 2378 | 3004.95 | 2589.1 | 8.87720774 | 0 | NearestNeighbor | LinKernighanHeuristic |
| gil262 | 2378 | 3004.95 | 3004.95 | 26.3645921 | 0 | NearestNeighbor | OptK |
| gil262 | 2378 | 3004.95 | 2555.63 | 7.46972246 | 4073 | NearestNeighbor | OptOr |
| gil262 | 2378 | 3004.95 | 2655.19 | 11.656434 | 9742 | NearestNeighbor | Opt3 |
| kroA100 | 21282 | 26856.39 | 22955.91 | 7.86537919 | 0 | NearestNeighbor | LinKernighanHeuristic |
| kroA100 | 21282 | 26856.39 | 26856.39 | 26.19298 | 0 | NearestNeighbor | OptK |
| kroA100 | 21282 | 26856.39 | 23647.64 | 11.1156846 | 94 | NearestNeighbor | OptOr |
| kroA100 | 21282 | 26856.39 | 23619.16 | 10.9818626 | 340 | NearestNeighbor | Opt3 |
| kroA150 | 26524 | 33609.87 | 28970.17 | 9.22247776 | 0 | NearestNeighbor | LinKernighanHeuristic |
| kroA150 | 26524 | 33609.87 | 33609.87 | 26.7149374 | 0 | NearestNeighbor | OptK |
| kroA150 | 26524 | 33609.87 | 29920.11 | 12.8039134 | 464 | NearestNeighbor | OptOr |
| kroA150 | 26524 | 33609.87 | 30089.94 | 13.4442015 | 917 | NearestNeighbor | Opt3 |
| kroA200 | 29368 | 35798.41 | 30056.67 | 2.34496731 | 1 | NearestNeighbor | LinKernighanHeuristic |
| kroA200 | 29368 | 35798.41 | 35798.41 | 21.8959752 | 0 | NearestNeighbor | OptK |
| kroA200 | 29368 | 35798.41 | 31693.18 | 7.91739308 | 1510 | NearestNeighbor | OptOr |
| kroA200 | 29368 | 35798.41 | 32434.45 | 10.4414669 | 2076 | NearestNeighbor | Opt3 |
| kroB100 | 22141 | 29155.04 | 22774.01 | 2.85899463 | 0 | NearestNeighbor | LinKernighanHeuristic |
| kroB100 | 22141 | 29155.04 | 29155.04 | 31.6789666 | 0 | NearestNeighbor | OptK |
| kroB100 | 22141 | 29155.04 | 25830.87 | 16.6653268 | 60 | NearestNeighbor | OptOr |
| kroB100 | 22141 | 29155.04 | 25203.76 | 13.8329795 | 293 | NearestNeighbor | Opt3 |
| kroB150 | 26130 | 32825.75 | 27993.86 | 7.13302717 | 1 | NearestNeighbor | LinKernighanHeuristic |
| kroB150 | 26130 | 32825.75 | 32825.75 | 25.6247608 | 0 | NearestNeighbor | OptK |
| kroB150 | 26130 | 32825.75 | 28440.23 | 8.84129353 | 393 | NearestNeighbor | OptOr |
| kroB150 | 26130 | 32825.75 | 28421.91 | 8.77118255 | 1377 | NearestNeighbor | Opt3 |
| kroB200 | 29437 | 36981.59 | 32598.09 | 10.7384924 | 1 | NearestNeighbor | LinKernighanHeuristic |
| kroB200 | 29437 | 36981.59 | 36981.59 | 25.6296158 | 0 | NearestNeighbor | OptK |
| kroB200 | 29437 | 36981.59 | 32583.16 | 10.6877739 | 1255 | NearestNeighbor | OptOr |
| kroB200 | 29437 | 36981.59 | 34531.31 | 17.3058056 | 2766 | NearestNeighbor | Opt3 |
| kroC100 | 20749 | 26327.36 | 22533.73 | 8.60152297 | 0 | NearestNeighbor | LinKernighanHeuristic |
| kroC100 | 20749 | 26327.36 | 26327.36 | 26.8849583 | 0 | NearestNeighbor | OptK |
| kroC100 | 20749 | 26327.36 | 23282.8 | 12.2116729 | 96 | NearestNeighbor | OptOr |
| kroC100 | 20749 | 26327.36 | 24488.3 | 18.0215914 | 196 | NearestNeighbor | Opt3 |
| kroD100 | 21294 | 26950.46 | 22804.97 | 7.09575467 | 0 | NearestNeighbor | LinKernighanHeuristic |
| kroD100 | 21294 | 26950.46 | 26950.46 | 26.563633 | 0 | NearestNeighbor | OptK |
| kroD100 | 21294 | 26950.46 | 24263.6 | 13.9457124 | 101 | NearestNeighbor | OptOr |
| kroD100 | 21294 | 26950.46 | 25110.89 | 17.9247206 | 292 | NearestNeighbor | Opt3 |
| kroE100 | 22068 | 27587.19 | 24312.11 | 10.1690683 | 0 | NearestNeighbor | LinKernighanHeuristic |
| kroE100 | 22068 | 27587.19 | 27587.19 | 25.0099239 | 0 | NearestNeighbor | OptK |
| kroE100 | 22068 | 27587.19 | 25282.07 | 14.5643919 | 64 | NearestNeighbor | OptOr |
| kroE100 | 22068 | 27587.19 | 24711.36 | 11.9782491 | 244 | NearestNeighbor | Opt3 |
| lin105 | 14379 | 20362.76 | 16199.7 | 12.6622157 | 1 | NearestNeighbor | LinKernighanHeuristic |
| lin105 | 14379 | 20362.76 | 20362.76 | 41.6145768 | 0 | NearestNeighbor | OptK |
| lin105 | 14379 | 20362.76 | 16832.08 | 17.0601572 | 137 | NearestNeighbor | OptOr |
| lin105 | 14379 | 20362.76 | 18306.7 | 27.3155296 | 234 | NearestNeighbor | Opt3 |
| pr107 | 44303 | 50519.13 | 45252.18 | 2.14247342 | 0 | NearestNeighbor | LinKernighanHeuristic |
| pr107 | 44303 | 50519.13 | 50519.13 | 14.030946 | 0 | NearestNeighbor | OptK |
| pr107 | 44303 | 50519.13 | 46761.18 | 5.5485633 | 94 | NearestNeighbor | OptOr |
| pr107 | 44303 | 50519.13 | 48219.83 | 8.841004 | 189 | NearestNeighbor | Opt3 |
| pr124 | 59030 | 71803.15 | 60494.49 | 2.48092495 | 0 | NearestNeighbor | LinKernighanHeuristic |
| pr124 | 59030 | 71803.15 | 71803.15 | 21.6384042 | 0 | NearestNeighbor | OptK |
| pr124 | 59030 | 71803.15 | 62971.32 | 6.6768084 | 116 | NearestNeighbor | OptOr |
| pr124 | 59030 | 71803.15 | 64692.44 | 9.5924784 | 334 | NearestNeighbor | Opt3 |
| pr136 | 96772 | 120777.86 | 106006.85 | 9.54289464 | 0 | NearestNeighbor | LinKernighanHeuristic |
| pr136 | 96772 | 120777.86 | 120777.86 | 24.8066176 | 0 | NearestNeighbor | OptK |
| pr136 | 96772 | 120777.86 | 105694.49 | 9.22011532 | 205 | NearestNeighbor | OptOr |
| pr136 | 96772 | 120777.86 | 112112.01 | 15.851703 | 630 | NearestNeighbor | Opt3 |
| pr144 | 58537 | 61650.72 | 61243.8 | 4.62408391 | 0 | NearestNeighbor | LinKernighanHeuristic |
| pr144 | 58537 | 61650.72 | 61650.72 | 5.31923399 | 0 | NearestNeighbor | OptK |
| pr144 | 58537 | 61650.72 | 59368.57 | 1.42058869 | 99 | NearestNeighbor | OptOr |
| pr144 | 58537 | 61650.72 | 61650.72 | 5.31923399 | 196 | NearestNeighbor | Opt3 |
| pr152 | 73682 | 85702.95 | 77862.01 | 5.67304091 | 0 | NearestNeighbor | LinKernighanHeuristic |
| pr152 | 73682 | 85702.95 | 85702.95 | 16.3146359 | 0 | NearestNeighbor | OptK |
| pr152 | 73682 | 85702.95 | 77329.44 | 4.95024565 | 414 | NearestNeighbor | OptOr |
| pr152 | 73682 | 85702.95 | 83066.22 | 12.7361092 | 722 | NearestNeighbor | Opt3 |
| pr226 | 80369 | 95713.92 | 83464.12 | 3.85113663 | 0 | NearestNeighbor | LinKernighanHeuristic |
| pr226 | 80369 | 95713.92 | 95713.92 | 19.0930832 | 0 | NearestNeighbor | OptK |
| pr226 | 80369 | 95713.92 | 84584.42 | 5.24508206 | 1411 | NearestNeighbor | OptOr |
| pr226 | 80369 | 95713.92 | 89448.07 | 11.2967313 | 6626 | NearestNeighbor | Opt3 |
| pr264 | 49135 | 58022.86 | 52848.89 | 7.55854279 | 1 | NearestNeighbor | LinKernighanHeuristic |
| pr264 | 49135 | 58022.86 | 58022.86 | 18.0886537 | 0 | NearestNeighbor | OptK |
| pr264 | 49135 | 58022.86 | 54624.35 | 11.1719752 | 1857 | NearestNeighbor | OptOr |
| pr264 | 49135 | 58022.86 | 57779.04 | 17.592429 | 4006 | NearestNeighbor | Opt3 |
| pr299 | 48191 | 62523.38 | 53507.02 | 11.0311469 | 1 | NearestNeighbor | LinKernighanHeuristic |
| pr299 | 48191 | 62523.38 | 62523.38 | 29.7407815 | 0 | NearestNeighbor | OptK |
| pr299 | 48191 | 62523.38 | 54630.43 | 13.3623083 | 6590 | NearestNeighbor | OptOr |
| pr299 | 48191 | 62523.38 | 53139.79 | 10.2691166 | 16131 | NearestNeighbor | Opt3 |
| rat195 | 2323 | 2761.96 | 2489.73 | 7.17735687 | 1 | NearestNeighbor | LinKernighanHeuristic |
| rat195 | 2323 | 2761.96 | 2761.96 | 18.8962548 | 0 | NearestNeighbor | OptK |
| rat195 | 2323 | 2761.96 | 2536.27 | 9.18080069 | 868 | NearestNeighbor | OptOr |
| rat195 | 2323 | 2761.96 | 2529.52 | 8.89022815 | 2508 | NearestNeighbor | Opt3 |
| rd100 | 7910 | 9941.16 | 8553.76 | 8.13855879 | 0 | NearestNeighbor | LinKernighanHeuristic |
| rd100 | 7910 | 9941.16 | 9941.16 | 25.6783818 | 0 | NearestNeighbor | OptK |
| rd100 | 7910 | 9941.16 | 8739.84 | 10.491024 | 112 | NearestNeighbor | OptOr |
| rd100 | 7910 | 9941.16 | 9019.72 | 14.02933 | 244 | NearestNeighbor | Opt3 |
| ts225 | 126643 | 155806.65 | 133600.49 | 5.49378173 | 1 | NearestNeighbor | LinKernighanHeuristic |
| ts225 | 126643 | 155806.65 | 155806.65 | 23.0282369 | 0 | NearestNeighbor | OptK |
| ts225 | 126643 | 155806.65 | 142404.61 | 12.4457017 | 841 | NearestNeighbor | OptOr |
| ts225 | 126643 | 155806.65 | 147964.32 | 16.8357667 | 3254 | NearestNeighbor | Opt3 |
| tsp225 | 3916 | 4786.42 | 4163.03 | 6.30822268 | 1 | NearestNeighbor | LinKernighanHeuristic |
| tsp225 | 3916 | 4786.42 | 4786.42 | 22.2272727 | 0 | NearestNeighbor | OptK |
| tsp225 | 3916 | 4786.42 | 4249.55 | 8.51762002 | 1809 | NearestNeighbor | OptOr |
| tsp225 | 3916 | 4786.42 | 4335.05 | 10.7009704 | 3256 | NearestNeighbor | Opt3 |
| u159 | 42080 | 53737.89 | 48204.44 | 14.5542776 | 1 | NearestNeighbor | LinKernighanHeuristic |
| u159 | 42080 | 53737.89 | 53737.89 | 27.7041112 | 0 | NearestNeighbor | OptK |
| u159 | 42080 | 53737.89 | 47786.81 | 13.5618108 | 440 | NearestNeighbor | OptOr |
| u159 | 42080 | 53737.89 | 47803.96 | 13.6025665 | 1433 | NearestNeighbor | Opt3 |

Tabela : Resultados resumidos

## 5.1 Qualidade das Soluções

A solução inicial obtida pelo Nearest Neighbor apresenta, em média, cerca de 25% de desvio face à solução ótima, variando entre 12% (instâncias mais fáceis, como eil101) e quase 31% (por exemplo, lin105). O Lin-Kernighan Heuristic destaca-se por reduzir esse desvio para valores médios em torno de 5%, chegando a menos de 3% em instâncias como kroA200 e pr107, o que demonstra a sua eficácia em escapar de ótimos locais (TabelaResultadosResumida).

O Or-Opt melhora a solução inicial em cerca de 7–11% de desvio médio, mas sem atingir a consistência do Lin-Kernighan. Em várias instâncias (por ex. bier127, pr144) o Or-Opt até supera ligeiramente o 3-Opt em termos de qualidade.

O 3-Opt, embora explore uma vizinhança maior que o Or-Opt, alcança desvio médio superior (cerca de 10–12%), indicando que o seu ganho marginal face ao Or-Opt não compensa a maior sobrecarga computacional.

O método genérico OptK não produziu melhoria em nenhuma instância (solução final igual à inicial), sugerindo que a parametrização atual ou a implementação necessitam de revisão (TabelaResultadosResumida).

Adicionalmente, testámos variantes do 3-Opt com estratégias distintas: First Improvement e Best Improvement, aplicadas a 6 instâncias (eil101, bier127, ch130, ch150, d198, a280). Observou-se que, embora o Best Improvement consiga ligeiramente melhores resultados em algumas instâncias (até -0.63% no caso de a280), o tempo de execução é significativamente superior — por vezes, 5 a 10 vezes mais lento. Em casos extremos como d198, o Best Improvement demorou mais de 80 segundos, enquanto o First Improvement atingiu resultados praticamente equivalentes em apenas 6 segundos. Esta diferença é ainda mais marcante em instâncias pequenas como eil101, onde o Best Improvement levou quase 20 vezes mais tempo por uma melhoria mínima. Assim, conclui-se que a versão First Improvement oferece um equilíbrio mais eficaz entre tempo e qualidade, sendo preferível em cenários onde o tempo de resposta é um critério relevante.

## 5.2 Tempo Computacional

O Lin-Kernighan Heuristic oferece o melhor compromisso entre qualidade e tempo: gera soluções até 5× melhores que a heurística construtiva de base, mantendo o custo computacional quase igual.

Or-Opt pode ser usado como etapa adicional quando se pretende um aperfeiçoamento extra sem incorrer nos custos do 3-Opt, sobretudo em instâncias de tamanho moderado.

O 3-Opt, apesar de explorar uma vizinhança mais ampla, apresenta retornos decrescentes em qualidade relativos ao aumento de tempo, sendo indicado apenas quando se dispõe de recursos computacionais e pouco dinamismo na aplicação.

Importa ainda referir que foram desenvolvidas versões de alguns destes algoritmos em Python, e verificou-se que o tempo de execução nestas implementações era drasticamente superior — em certos casos, algoritmos que em java eram resolvidos em minutos levavam várias horas ou mesmo dias em Python, sobretudo no caso das heurísticas de melhoria. Este contraste evidencia a importância da eficiência da linguagem escolhida quando se pretende aplicar estas técnicas em cenários práticos.

## 5.3 Conclusões Parciais e Recomendações

Devido à sua eficácia comprovada, o Lin-Kernighan deve ser priorizado como principal heurística de pós-processamento, enquanto o Or-Opt pode ser incluído em cenários intermédios para proporcionar ganhos rápidos adicionais; o 3-Opt deverá ser reservado para estudos offline ou instâncias críticas, em que cada fração percentual de melhoria justifique o tempo extra, e é fundamental rever a implementação do OptK para assegurar que ele explora efetivamente as vizinhanças k-Opt, em vez de retornar sistematicamente à solução inicial.

# 6. Conclusões e Trabalho Futuro

O trabalho desenvolvido permitiu não só aprofundar o conhecimento teórico sobre o Problema do Caixeiro Viajante, como também experimentar e validar diferentes abordagens algorítmicas. As heurísticas construtivas demonstraram ser ferramentas valiosas para gerar soluções iniciais rapidamente, enquanto os métodos de melhoria, em especial o Lin-Kernighan, revelaram-se eficazes na obtenção de soluções significativamente mais próximas do ótimo.

Entre os pontos fortes do projeto destacam-se a modularidade do código, a clareza na comparação entre algoritmos e a utilização de múltiplas instâncias de teste com dados reais. No entanto, seria desejável explorar de forma mais aprofundada o papel das estruturas de dados avançadas e investigar estratégias de paralelização para acelerar as heurísticas mais exigentes.

Como trabalho futuro, propõe-se:

* A integração de uma interface gráfica que permita visualizar os percursos obtidos em tempo real.
* A implementação de algoritmos híbridos que combinem características de diferentes heurísticas.
* A análise de instâncias assimétricas e com restrições adicionais, como janelas temporais ou capacidades logísticas.

## Referências Bibliográficas

[1] R. M. Karp, "Reducibility among combinatorial problems," Complexity of Computer Computations, New York: Plenum, 1972, pp. 85–103.

[2] M. Held and R. M. Karp, "A dynamic programming approach to sequencing problems," Journal of the Society for Industrial and Applied Mathematics, vol. 10, no. 1, pp. 196–210, 1962.

[3] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," Operations Research, vol. 21, no. 2, pp. 498–516, Mar. 1973.

[4] M. Dorigo and L. M. Gambardella, "Ant colonies for the traveling salesman problem," Biosystems, vol. 43, no. 2, pp. 73–81, 1997.

[5] F. Glover, "Tabu Search—Part I," ORSA Journal on Computing, vol. 1, no. 3, pp. 190–206, 1989.

[6] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," Science, vol. 220, no. 4598, pp. 671–680, 1983.

# Referências WWW

[01] <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>

Página principal da TSPLIB. Aqui podemos encontrar conjuntos de dados clássicos para problemas de otimização como o TSP, usados para benchmarking de algoritmos.

[02] <https://en.wikipedia.org/wiki/Travelling_salesman_problem>

Página da Wikipédia dedicada ao Problema do Caixeiro Viajante, com uma introdução geral, abordagens de resolução e aplicações práticas.

[03] <https://www.sciencedirect.com/topics/computer-science/nearest-neighbor-algorithm>

Descrição do algoritmo Vizinho Mais Próximo, incluindo vantagens, limitações e aplicações.

[04] <https://www.geeksforgeeks.org/2-opt-algorithm-for-traveling-salesman-problem/>

Artigo explicativo sobre a heurística de melhoria 2-Opt aplicada ao TSP, com exemplos de implementação.

[05] <https://www.sciencedirect.com/science/article/pii/S0957417419303784>

Estudo científico sobre heurísticas e meta-heurísticas modernas aplicadas a problemas de roteamento como o TSP.