UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
DEPARTAMENTO DE CIÊNCIAS E TECNOLOGIA
CURSO DE BACHARELADO EM CIÊNCIAS E TECNOLOGIA
TRABALHO DE CONCLUSÃO DE CURSO (2025)

APLICAÇÃO DE ALGORITMOS GENÉTICOS PARA CALCULAR ROTAS PARA DISTRIBUIÇÃO DE ENTREGAS

APPLICATION OF GENETIC ALGORITHMS TO CALCULATE ROUTES FOR DELIVERY DISTRIBUTION

Anderson Esmael da Silva
Rodrigo Toledo Texeira Camara

RESUMO

O problema do caixeiro viajante (TSP, na sigla em inglês) tem como finalidade determinar o melhor caminho que um viajante deve percorrer para visitar diversos pontos e retornar ao ponto de partida. Este problema é considerado de alta complexidade computacional, conforme destacado por diversos autores (GREFENSTETTE, 2014; LARRANAGA et al., 1999, p. 2). Dentre as abordagens para solucionar o TSP, destaca-se o uso de algoritmos genéticos, uma classe de algoritmos inspirada na teoria da evolução de Charles Darwin e desenvolvida inicialmente por John Holland em 1975 (HOLLAND, 1975; GOLDBERG, 1989). Este trabalho tem como objetivo documentar a implementação de um software que calcula a melhor rota de entrega utilizando algoritmo genético. Para exemplificação, os pontos de entrega foram selecionados no entorno do campus da UFERSA de Angicos- RN, visando demonstrar a aplicabilidade da solução em um contexto real.

Palavras-chave: Algoritmo Genético, caixeiro viajante, software, otimização.

ABSTRACT

The Traveling Salesman Problem (TSP) aims to determine the optimal path a traveler should take to visit various points and return to the starting point. This problem is considered to be of high computational complexity, as highlighted by several authors (GREFENSTETTE, 2014; LARRANAGA et al., 1999, p. 2). Among the approaches to solve the TSP, the use of genetic algorithms stands out, a class of algorithms inspired by Charles Darwin's theory of evolution and initially developed by John Holland in 1975 (HOLLAND, 1975; GOLDBERG, 1989). This work aims to document the implementation of a software that calculates the best delivery route using a genetic algorithm. For illustration, the delivery points were selected around the university campus of UFERSA, in Angicos- RN, Brazil, aiming to demonstrate the applicability of the solution in a real-world context.

Keywords: genetic algorithm, traveling salesman, software, optimization.

1 INTRODUÇÃO

Este trabalho aborda uma forma de resolver o Problema do Caixeiro Viajante, um problema considerado clássico na teoria dos grafos. O problema envolve a busca de um melhor caminho que um viajante deve fazer para visitar diversos pontos e retornar para o começo (Enami et al., 2019, p. 4). Este artigo estuda a otimização de rotas para melhorias na eficiência nos processos de entrega.

A otimização de rotas é um dos grandes desafios empresariais devido às condições geográficas para as empresas desses setores, do qual o mesmo influencia no custo final do produto, ou seja, no quanto ele será vendido para os consumidores (Dutra, 2023, p. 2).

Dessa forma, a busca por rotas mais curtas e eficientes permite minimizar o tempo e reduzir os custos, melhorando a satisfação do cliente. Proporcionando o uso do tempo e do espaço, permitindo que o produto seja colocado na hora e no lugar certo. Após os avanços tecnológicos, a interação homem-máquina no ambiente logístico tornou-se um fator chave para enfrentar os desafios diários de um mundo em constante evolução (Campos; Compilada, 2013, p. 1).

Diante disso, o objetivo deste trabalho é a implementação de um *software* focado em otimizar rotas, utilizando algoritmos genéticos.

Para implementar este *software* utilizaremos uma classe de algoritmos conhecida como Genetic Algorithms (GAs), ou “Algoritmos Genéticos” (AGs) em português (tradução nossa).

Holland (1975) explicou esses algoritmos, tendo como característica principal a simulação de processos evolutivos naturais, baseando-se em conceitos como seleção, cruzamento e mutação para otimizar soluções dentro de um espaço de busca. Esta classe de algoritmo tem, portanto, semelhanças com a teoria da evolução proposta por Charles Darwin.

De acordo com Goldberg (1989, p .20) estes algoritmos são métodos inspirados na teoria do darwinismo, baseados nos princípios da seleção natural e na reprodução genética das espécies. Ele aponta que, apesar desses algoritmos serem bastante robustos, eles são particularmente eficientes e eficazes para problemas que envolvem espaços de busca complexos.

2 DESENVOLVIMENTO

2.1 Algoritmos Genéticos

Os Algoritmos Genéticos representam uma abordagem de otimização e busca inspirada nos mecanismos da evolução biológica, como seleção natural, mutação e recombinação. Desenvolvidos por John Holland e sua equipe na Universidade de Michigan nas décadas de 1960 a 1970, os AGs foram inicialmente concebidos para simular processos adaptativos em sistemas complexos (Correia, 2004, p. 1).

Holland desenvolveu os algoritmos genéticos como um modelo computacional para analisar como os sistemas podem adaptar-se de maneira eficiente, utilizando os fundamentos da seleção natural, recombinação e mutação, conforme descrito por Charles Darwin. Inicialmente, Holland não estava interessado apenas em resolver problemas, mas sim, em entender como esses sistemas podem evoluir ao longo do tempo, buscando soluções otimizadas (Mitchell, 1998, p. 7).

Em 1975, ele formalizou sua ideia, marcando um grande passo na computação. Holland sugeriu um algoritmo que usa uma população de soluções. Esse método inclui operações como cruzamento, inversão e mutação. Isso difere de abordagens anteriores que usavam só mutação ou mudanças limitadas. A nova forma gerou muitas soluções para problemas difíceis. Sua teoria apoiava-se no fundamento de “esquemas”, que são padrões ou estruturas que, através da evolução, tornam-se mais fortes e prevalentes nas soluções para um problema (Mitchell, 1998, p. 7).

Nesta seção, apresentamos os principais elementos que compõem os algoritmos genéticos, fundamentais para resolver problemas de busca e otimização, como a Função Avaliadora de Aptidão e os Operadores genéticos (*Seleção natural, Cruzamento e Mutação*) conforme descrito por Correia (2004, p. 4).

2.2 Função *Fitness*

A *função fitness* tem como propósito avaliar os indivíduos com base na função objetivo, fornecendo parâmetros que permitam interpretar o desempenho de cada indivíduo diante do problema considerado. Um exemplo muito comum ocorre nos problemas de maximização, em que os indivíduos mais aptos apresentam valores numéricos mais elevados associados à *função objetivo*. Por outro lado, nos problemas de minimização, os indivíduos mais aptos exigem os menores valores numéricos relacionados à sua *função fitness.* Esses padrões são amplamente empregados para determinar a *aptidão relativa* dos indivíduos. Geralmente, utiliza-se a *função fitness* para

converter o valor da *função objetivo* em uma medida de *fitness relativo*, conforme descrito por Correia (2004) como

$$F(x) = g(f(x))$$

.

Esta equação representa a estrutura básica de uma função objetivo transformada em uma medida do *fitness relativo*, onde *f* representa a *função objetivo*, *g* realiza a conversão dos valores para números não negativos, e *F* corresponde à *aptidão relativa* resultante. Este sistema é aplicado quando a *função objetivo* está relacionada a um problema de minimização. Contudo, em problemas de maximização, a *função objetivo* pode gerar valores negativos.

Normalmente, a função *fitness* está associada ao número de descendentes que os indivíduos podem gerar na próxima geração. Essas transformações são utilizadas na função avaliadora, sendo proporcionais à aptidão individual *F(x)*, que é calculada com base no desempenho de cada indivíduo e na medida primária de *fitness f(x)*, em relação ao desempenho global da população.

A equação

$$F(x_i) = \frac{f(x_i)}{\sum_{i=1}^{NInd} f(x_i)}$$

indica que *Nind* caracteriza o tamanho da população, enquanto *xi* corresponde ao fenótipo do indivíduo avaliado *i*. Existem duas abordagens principais para aprimorar a *função fitness*, descritas por Correia (2004) como a seguir:

- Escalonamento (Scaling): Trata-se de uma técnica empregada para ajustar a *função fitness* considerando os níveis de desempenho de todos os indivíduos da população. O escalonamento garante que os valores de *aptidão* sejam sempre não negativos, regulando a distribuição de aptidão entre os membros da população. Além disso, essa abordagem limita a frequência com que cada indivíduo é selecionado para a reprodução, promovendo maior equilíbrio no processo evolutivo.
- Classificação (Ranking): Nesse método os indivíduos da população são organizados em ordem com base nos valores da *função objetivo*. Após essa ordenação, cada indivíduo recebe um valor de aptidão proporcional à sua posição

no ranking. Essa técnica reduz o impacto de valores extremos de aptidão, garantindo uma seleção mais equilibrada e incentivando a diversidade na população ao longo do processo evolutivo.

2.3 Operadores genéticos

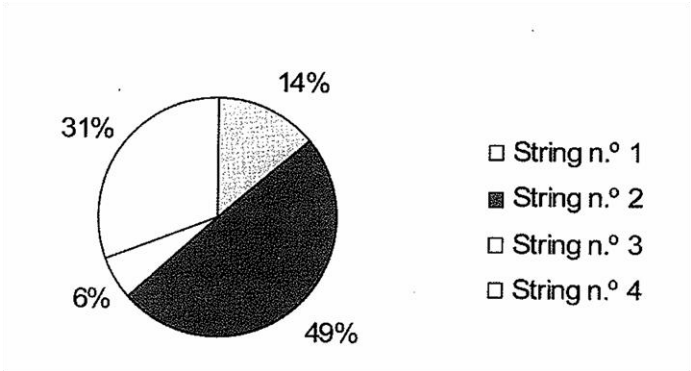
Os operadores genéticos atuam como ferramentas que promovem alterações na população ao longo das gerações. Essas mudanças são essenciais para garantir a diversidade das soluções, sobretudo aquelas mais eficientes, assegurando sua conservação e transmissão às gerações futuras. Além disso, esses operadores introduzem variações, elemento indispensável para manter o processo de adaptação em constante atividade, favorecendo a evolução constante das soluções. Há diferentes tipos de operadores genéticos, sendo os mais relevantes apresentados a seguir:

2.4 Seleção natural

Os operadores de seleção determinam quais os indivíduos serão selecionados para reprodução, priorizando aqueles mais adaptados ao problema, ou seja, os que apresentam maior *aptidão*. Esses indivíduos tornam-se os pais da próxima geração, assegurando que suas características sejam herdadas por seus filhos.

Um dos métodos de seleção mais conhecidos é o "Método da Roleta", desenvolvido por Holland.

Figura 1 – Método da roleta



Fonte: Correia (2004)

Nesse método, a probabilidade de um indivíduo ser selecionado é proporcional à sua *aptidão*, de modo que os mais bem adaptados possuem maior chance de gerar

descendentes. Essa abordagem assegura que características vantajosas tenham maior probabilidade de serem preservadas na população ao longo do tempo.

2.5 Cruzamento

Os operadores de cruzamento apresentam um papel fundamental nos algoritmos genéticos, promovendo a troca de informações genéticas entre os indivíduos selecionados como progenitores. Esse processo combina traços de diferentes indivíduos, disseminando características vantajosas e possibilitando melhorias nas soluções ao longo das gerações.

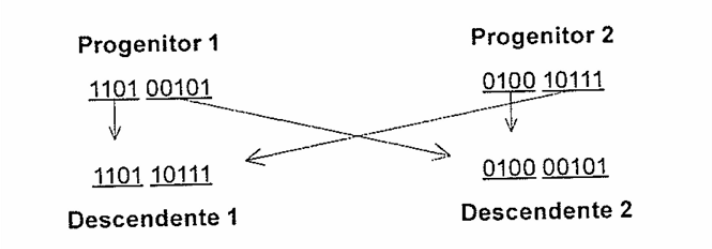
Através do processo de cruzamento, novas combinações de genes são criadas, o que pode resultar em soluções mais eficientes para o problema em análise. Especificamente, o operador de cruzamento une os genes de dois cromossomos progenitores, formando dois novos cromossomos na geração seguinte. Esses descendentes, em geral, apresentam maior aptidão em comparação aos seus antecessores, contribuindo para o fortalecimento da população.

Para que o cruzamento seja eficaz, é necessário ajustar a taxa de cruzamento, que determina a probabilidade de aplicação desse operador. Uma taxa bem calibrada equilibra a diversidade genética e a estabilidade das soluções, garantindo que o algoritmo evolua de forma consistente em direção a resultados mais eficientes. Uma taxa de cruzamento igual a 1.0 substitui todos os indivíduos, acelerando a geração de novas combinações, mas também pode eliminar indivíduos altamente adaptados. Por outro lado, taxas mais baixas ajudam a preservar soluções valiosas, embora possam tornar o algoritmo mais lento. Ainda segundo Correia (2004), encontrar o equilíbrio dessa taxa é fundamental para explorar novas possibilidades sem comprometer as melhores soluções geradas ao longo das gerações.

No cruzamento de um ponto, por exemplo, um único ponto é selecionado aleatoriamente ao longo da sequência do cromossomo, dividindo-o em dois segmentos. Os segmentos após esse ponto são trocados entre os pais, resultando em dois novos descendentes que combinam características combinadas de seus genitores. Esse método simples, mas eficiente, contribui para aumentar a diversidade da população e potencializar a adaptação às demandas do problema.

Conforme ilustrado na figura 2, caso o ponto de corte seja o 4° *bit*, os três primeiros *bits* permanecem inalterados em ambos os descendentes, enquanto os *bits* seguintes são trocados.

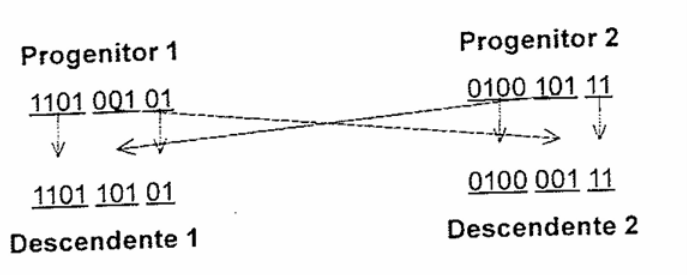
Figura 2 – Cruzamento de múltiplos pontos



Fonte: Correia (2004)

Na figura 3, é apresentado um exemplo de cruzamento de dois pontos, em que o 4° e 7° *bits* foram selecionados como pontos de corte. Nesse processo, as cadeias de *bits* de duas soluções são divididas nesses pontos, e as partes intermediárias são trocadas entre elas. A seção antes do 4° *bit* permanece inalterada, enquanto o segmento entre o 4° e o 7° *bit* é intercambiado, e a parte após o 7° *bit* é herdada da outra solução. Esse tipo de cruzamento tem como objetivo gerar novas combinações de soluções.

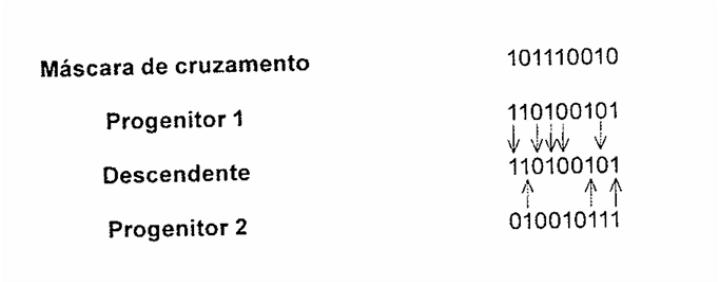
Figura 3 – Cruzamento entre dois pontos



Fonte: Correia (2004)

No cruzamento uniforme, cada gene dos descendentes é determinado a partir de um dos progenitores, com base em uma máscara de cruzamento gerada de forma aleatória. Essa máscara é composta por *bits*, em que cada *bit* define de qual progenitor será copiado o gene correspondente. Quando um *bit* da máscara é 1, o gene é obtido do primeiro progenitor; se for 0, o gene é herdado do segundo progenitor. Esse método de cruzamento promove uma combinação aleatória dos genes de ambos os pais, resultando em maior diversidade genética entre os descendentes, conforme ilustrado na figura 4.

Figura 4 – Cruzamento Uniforme



Fonte: Correia (2004)

2.6 Mutação

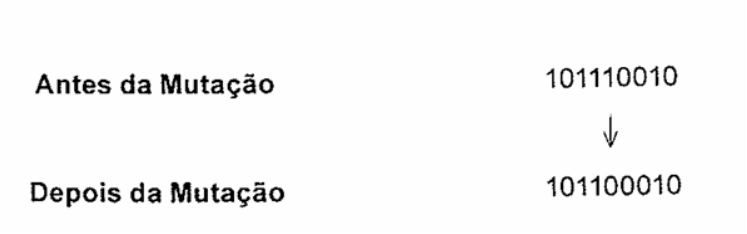
Os operadores de mutação são responsáveis por introduzir e manter a diversidade genética na população de soluções. Essa diversidade permite que os algoritmos explorem diferentes regiões do espaço de busca, evitando que fiquem presos em soluções locais.

A mutação altera aleatoriamente um ou mais componentes de uma solução da descendência, possibilitando a criação de novos elementos na população. Dessa forma, assegura-se que qualquer ponto do espaço de busca tenha uma probabilidade não nula de ser alcançado.

Quando a representação é binária, a mutação ocorre por meio da troca de valores de *bits*, ou seja, um valor 0 pode ser transformado em 1, ou vice-versa. Essa modificação, aparentemente simples, pode ter um impacto considerável, fazendo com que o cromossomo represente um ponto completamente distinto no espaço de busca. Isso amplia a capacidade do algoritmo de explorar novas soluções e descobrir resultados inovadores.

A aplicação do operador de mutação em um cromossomo é ilustrada na figura 5, que mostra a alteração específica do 5° *bit*. Essa mudança evidencia como uma pequena modificação pode influenciar significativamente o comportamento do algoritmo, reforçando a importância da mutação como ferramenta para diversificar as soluções e aprimorar o desempenho do processo de otimização.

Figura 5 – Operador Mutação



Fonte: Correia (2004)

Na figura 5, demonstra-se a aplicação do operador de mutação em um cromossomo representado por uma sequência binária de 8 *bits*. Antes da mutação, a sequência inicial era 101110010 e, após a mutação, a sequência foi alterada para 101100010. A única diferença entre as duas sequências está 5° *bit*, contando da esquerda para a direita, que antes da mutação era "1" e, após a mutação, foi alterado para "0".

Essa alteração representa o conceito de mutação genética nos algoritmos evolutivos, nos quais pequenas alterações aleatórias nos cromossomos contribuem para a manutenção da diversidade genética na população de soluções. Isso evita que o algoritmo fique preso em mínimos locais e amplia sua capacidade de encontrar soluções mais eficientes no espaço de busca.

3 METODOLOGIA

O presente trabalho consiste na implementação de um *software* para otimização de rotas utilizando Algoritmos Genéticos interativos, com o objetivo de calcular rotas eficientes para a distribuição de entregas em torno do *campus* da UFERSA-Angicos. A proposta visa estabelecer a menor distância total percorrida, resultando em uma logística mais eficaz. Este artigo documenta o levantamento teórico necessário e o processo de desenvolvimento do software, que foi desenvolvido na linguagem *Python* e executado no *Google Colab*. A escolha dessa plataforma proporcionou uma interação simples e funcional para o usuário, sem a necessidade de uma interface gráfica avançada. A metodologia foi dividida em etapas, conforme descrito a seguir:

3.1 Levantamento de Dados e Estruturação do Problema

Inicialmente, foi realizado o levantamento dos dados necessários para estudar o problema da entrega otimizada, incluindo a localização geográfica de cada ponto de entrega. Para isso, foi estabelecida uma estrutura de dados no código, permitindo que o usuário defina os pontos de entrega, insira as distâncias entre eles e, com base nesses dados, obtenha a melhor rota por meio do processo de otimização.

Os pontos de entrega foram armazenados em uma lista global chamada *"pontos"*, onde cada ponto é denotado como $p_2,...,p_n$. O ponto de origem, o qual o viajante deve retornar ao final das visitas, é denotado por p_1. As distâncias entre os pontos foram armazenadas em uma matriz simétrica, conforme a figura 6.

Figura 6 – Matriz Simétrica

$$\texttt{matrizdedistancia} = \begin{bmatrix} d_{11} & & \dots & d_{1n} \\ \vdots & d_{ij} & & \\ dn1 & & \dots & d_{nn} \end{bmatrix}, \qquad (3)$$

,

Fonte: Autoria própria (2025).

Cada elemento *dij* representa a distância entre os pontos *pi* e *pj*. Essa estrutura permite que, caso seja necessário alterar a lista dos pontos de entrega disponíveis, um desenvolvedor possa atualizar a matriz de distância de forma simples.

3.2 A Estrutura do Código

A estrutura do código foi projetada para ser simples e intuitiva, permitindo que o usuário interaja com o sistema sem a necessidade de conhecimentos técnicos avançados. A interação ocorre por meio de entradas e saídas de texto no console, seguindo um fluxo lógico e amigável. Abaixo estão os principais elementos da estrutura:

• Listagem dos Pontos de Entrega: Ao iniciar o programa, o usuário visualiza uma lista de todos os pontos de entrega disponíveis, cada um associado a um número. Essa listagem ajuda o usuário a identificar os locais que deseja incluir na rota.

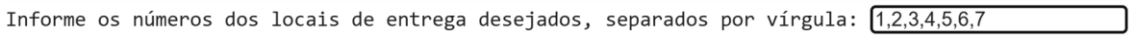
Figura 7 – Lista dos pontos de Entrega.

```
Bem-vindo ao Software de Otimização de Rotas com Algoritmos Genéticos!
Lista de Locais de Entrega Disponíveis:
1: Almoxarifado
2: Biblioteca
3: Bloco 1 de Aulas
4: Bloco 2 de Aulas
5: Bloco administrativo
6: Bloco 1 de Professores
7: Bloco 2 de Professores
8: Estacionamento
9: Garagem
10: Ginásio
11: Laboratórios 1
12: Laboratórios 2
13: Memorial Paulo Freire
14: Praça das Flores
15: Restaurante Universitário
16: Condomínio Canaã
17: Sala do Prof Maristélio Costa
```

Fonte: Autoria própria (2025).

• Seleção dos pontos de Entrega: O usuário é solicitado a informar os números dos locais de entrega desejados, separados por vírgula. O programa valida a entrada para garantir que todos os números sejam válidos e que pelo menos dois pontos sejam selecionados.

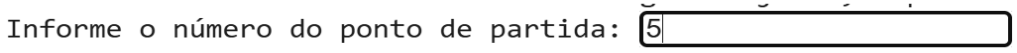
Figura 8 – Seleção dos pontos de Entrega.



Fonte: Autoria própria (2025).

• Escolha do Ponto de Partida: Após selecionar os pontos, o usuário deve informar o número do ponto de partida. O ponto de partida deve ser um dos pontos selecionados anteriormente.

Figura 9 – Escolha do Ponto de Partida.



Fonte: Autoria própria (2025).

• Exibição da Matriz de Distâncias: Após a seleção dos pontos, o programa exibe a matriz de distâncias entre os locais escolhidos, formatada de forma clara e organizada. Isso permite que o usuário visualize as distâncias entre os pontos antes de prosseguir com a otimização.

Figura 10 – Exibição da Matriz de Distâncias.

Matriz de Distâncias (em metros):

	Almoxarifa	Biblioteca	Bloco 1 de	Bloco 2 de	Bloco admi	Bloco 1 de	Bloco 2 de
Almoxarifa	0	350	550	290	500	140	290
Biblioteca	350	0	400	190	190	210	190
Bloco 1 de	550	400	0	400	400	400	400
Bloco 2 de	290	190	400	0	350	150	130
Bloco admi	500	190	400	350	0	350	350
Bloco 1 de	140	210	400	150	350	0	150
Bloco 2 de	290	190	400	130	350	150	0

Fonte: Autoria própria (2025).

• Execução do Algoritmo Genético: O programa executa o algoritmo genético automaticamente após a seleção dos pontos e exibe o progresso durante a execução. O usuário pode acompanhar o número de gerações e o tempo decorrido.

• Resultados da Otimização: Ao final da execução, o programa exibe a melhor rota encontrada, mostrando a sequência de pontos a serem visitados e a distância total percorrida.

Figura 11 – Resultados da Otimização.

```
Melhor Trajeto Calculado:
Bloco 2 de Aulas -> Almoxarifado (290 metros)
Almoxarifado -> Bloco 1 de Professores (140 metros)
Bloco 1 de Professores -> Bloco 1 de Aulas (400 metros)
Bloco 1 de Aulas -> Bloco administrativo (400 metros)
Bloco administrativo -> Biblioteca (190 metros)
Biblioteca -> Bloco 2 de Professores (190 metros)
Bloco 2 de Professores -> Bloco 2 de Aulas (130 metros)
Distância total Percorrida: 1740 metros
```

Fonte: Autoria própria (2025).

• Tratamento de Erro: A interface inclui mecanismos de tratamento de erros para garantir que o usuário insira dados válidos. Caso ocorra um erro, como a inserção de um número inválido, o programa exibe uma mensagem clara e solicita que o usuário tente novamente.

3.3 Implementação do Algoritmo genético

A implementação do algoritmo genético foi realizada utilizando a biblioteca DEAP (*Distributed Evolutionary Algorithms in Python*). A DEAP é uma biblioteca de código aberto que fornece ferramentas para a criação e execução de algoritmos evolucionários, como algoritmos genéticos, programação genética e estratégias de evolução. Ela foi escolhida por sua flexibilidade, facilidade de uso e suporte a diversas operações genéticas. O processo de implementação foi estruturado da seguinte forma:

• Representação dos indivíduos: Cada indivíduo (solução candidata) foi representado como uma lista de índices, onde cada índice corresponde a um ponto de entrega. A ordem dos índices define a rota a ser percorrida. Por exemplo, um indivíduo pode ser representado como [0,3,1,2], indicando a ordem em que os pontos devem ser visitados.

• Função Fitness: A função *fitness* calcula a distância total percorrida em uma rota, somando as distâncias entre os pontos consecutivos e incluindo o retorno ao ponto inicial,

o objetivo é minimizar essa distância. A função calcular distância total foi implementada para realizar esse cálculo, utilizando a matriz de distâncias como referência.

3.4 Operadores Genéticos

• Seleção: A seleção foi utilizada por torneio, onde os indivíduos com melhor fitness têm maior chance de serem selecionados para a próxima geração. A DEAP fornece a função *tools.selTournament,* que facilita a implementação desse operador.

• Cruzamento (*Crossover*): O operador de cruzamento escolhido foi o *OrderedCrossover (OX),* que preserva a ordem e a posição dos pontos na rota. Esse operador é particularmente útil para problemas de permutação, como o TSP. A DEAP oferece a função *tools.cxOrdered* para implementar esse tipo de cruzamento.

• Mutação: A mutação foi aplicada como uma taxa inicial de 10% utilizando o operador *ShuffleIndexes*, que reorganiza aleatoriamente parte da rota. A função to*ols.mutShuffleIndexes* da DEAP foi utilizada para implementar essa operação.

• Elitismo: Para garantir que as melhores soluções não sejam perdidas, foi implementado um mecanismo de elitismo, onde os dois melhores indivíduos de cada geração são automaticamente preservados para a próxima. Isso ajuda a manter a qualidade das soluções ao longo das gerações.

• Critérios de Parada: O algoritmo foi configurado para parar após 500 gerações ou se não houver melhorias no fitness por 10 gerações consecutivas. Além disso, um tempo máximo de execução de 60 segundos foi definido para evitar longos tempos de processamento. Esses critérios garantem que o algoritmo não fique preso em soluções subótimas e que o tempo de execução seja controlado.

3.5 Execução do Algoritmo

O algoritmo foi executado com uma população inicial de 100 indivíduos. A cada geração, os operadores genéticos foram aplicados para criar uma nova população, mantendo os melhores indivíduos (elitismo). O processo foi repetido até que um dos critérios de parada fosse atingido. Durante a execução, o algoritmo monitora o tempo decorrido e o número de gerações sem melhoria, garantindo que a solução seja encontrada de forma eficiente.

3.6 Visualização dos Resultados

Para validar a solução, foram realizados testes com diferentes quantidades de pontos de entrega (a saber, de 2 a 17 pontos). A aplicabilidade da solução foi demonstrada em um cenário real, utilizando pontos de entrega no entorno do *campus* da UFERSA na cidade de Angicos-RN, conforme foram mostrados anteriormente, na figura 7.

Para cada quantidade de pontos, o teste foi feito da seguinte maneira: primeiro tomam-se os pontos de forma aleatória. Destes pontos, toma-se uma rota aleatória que passa por todos eles. Em seguida, o *software* foi utilizado para, a partir dos pontos tomados, calcular a rota otimizada. As distâncias das rotas foram comparadas.

Os resultados obtidos, descritos na Tabela 1, foram consistentes e mostraram que o algoritmo é capaz de encontrar rotas melhores em um tempo razoável.

Tabela 1 – Testes de Eficácia do *Software*

Qtd de pontos	Rota aleatória	Distância da rota aleatória (metros)	Distância otimizada (metros)	Redução (%)	Tempo de execução (segundos)
2	14, 3	800	800	0	<1
3	11, 9, 15	750	750	0	<1
4	7, 2, 13, 3	1500	1300	13,3	<1
5	1, 11, 7, 9, 6	1200	900	25	<1
6	3, 4, 6, 11, 16, 10	3400	3000	11,7	<1
7	1, 13, 14, 15, 9, 2, 10	1800	1300	27,7	<1
8	15, 17, 1, 5, 8, 10, 9, 3	5400	4100	24	<1
9	1, 17, 9, 13, 14, 12, 4, 3, 7	5200	3800	26,9	<1
10	4, 9, 5, 8, 17, 16, 3, 1, 11, 15	4800	3980	17	<1
11	3, 8, 7, 2, 16, 14, 1, 4, 9, 12, 10	5100	3600	29,4	<1
12	5, 12, 7, 17, 11, 2, 6, 16, 13, 14, 8, 3	7100	3700	47,8	<1
13	2, 4, 15, 17, 10, 14, 5, 3, 11, 1, 9, 8, 7	6200	4050	34,6	<1
14	11, 1, 5, 9, 2, 12, 15, 16, 13, 7, 6, 3, 14, 10	6300	3750	40,4	<1
15	15, 1, 11, 16, 6, 4, 13, 5, 8, 2, 12, 7, 3, 17, 9	7200	3900	45,8	<1
16	6, 10, 3, 13, 7, 14, 8, 5, 12, 9, 11, 4, 1, 17, 15, 2	7200	4100	43	<1

Qtd de pontos	Rota aleatória	Distância da rota aleatória (metros)	Distância otimizada (metros)	Redução (%)	Tempo de execução (segundos)
17	15, 10, 7, 8, 1, 16, 13, 5, 12, 2, 9, 17, 14, 3, 4, 6, 11	9000	4000	55,5	<1

Fonte: Autoria própria (2025)

4 CONCLUSÃO

O desenvolvimento deste trabalho permitiu a criação de uma solução robusta para o problema de otimização de rotas, evidenciando a eficácia dos algoritmos genéticos na resolução de desafios complexos na área de logística e distribuição. A utilização da biblioteca DEAP facilitou a implementação e o ajuste dos operadores genéticos, demonstrando sua flexibilidade e poder computacional. Os resultados alcançados foram consistentes e apresentam potencial para aplicação em cenários reais, com adaptações específicas conforme as necessidades de cada contexto. Além disso, a metodologia empregada pode ser estendida para outros problemas de otimização combinatória, como o roteamento de veículos ou alocação de recursos.

O *software* desenvolvido mostrou-se capaz de calcular rotas otimizadas em um tempo consideravelmente rápido (menos de 1 segundo), desde que a conexão com a internet esteja estável. Essa rapidez é um diferencial importante para aplicações práticas, onde a agilidade na tomada de decisões logísticas é essencial. No entanto, observou-se que, à medida que o número de pontos selecionados aumenta, o desempenho do *software* é reduzido, principalmente devido ao acúmulo de memória na plataforma *Google Colab*. Essa limitação pode impactar a escalabilidade do sistema em cenários com grande volume de dados, sugerindo a necessidade de otimização futuras no código para melhorar a eficiência computacional.

Para trabalhos futuros, sugere-se a adoção de uma medida de distância que incorpore a dificuldade de atravessar determinados caminhos, sejam em termos de tempo ou consumo de combustível. Essa abordagem permitiria uma representação mais realista das condições de deslocamento, aumentando a precisão e a utilidade do sistema. Outra melhoria proposta é a implementação do *software* de modo completamente *offline*, eliminando a dependência de uma conexão estável com a internet e ampliando sua aplicabilidade em locais com infraestrutura limitada.

Além disso, recomenda-se a exploração de outras linguagens de programação, como *Java* ou *C++*, que podem oferecer maior desempenho em cenários com grande

volume de dados ou restrições de *hardware*. Outra sugestão é o desenvolvimento de um aplicativo interativo utilizando o *Folium*[1], que proporcionaria uma interface mais amigável e visualmente intuitiva para os usuários finais, facilitando a visualização e interação com as rotas otimizadas.

Por fim, uma melhoria relevante seria a implementação de funções no código para leitura de arquivos em formato `txt` contendo pontos pré-selecionados. Isso permitiria ao usuário carregar grandes conjuntos de dados de forma mais eficiente, sem a necessidade de inserir manualmente cada ponto. Essa funcionalidade não apenas melhora a usabilidade do software, mas também otimiza o processo de entrada de dados, especialmente em cenários com um grande número de localizações.

Este trabalho não apenas validou a eficácia dos algoritmos genéticos na otimização de rotas, mas também abriu caminho para novas pesquisas e aplicações práticas. Acredita-se que as contribuições aqui apresentadas possam servir como base para o desenvolvimento de soluções ainda mais robustas e adaptáveis, promovendo avanços significativos na área de logística e distribuição.

1 *Folium* é uma biblioteca *Python* que permite criar mapas interativos.

REFERÊNCIAS

CAMPOS, G; COMPILADA, A. **Logística**. Volume único. Disponível em: <https://estrategiaycompetitividad.wordpress.com/wp-content/uploads/2016/12/logistica-nuevo-mundo-nov-dic-17-mundo-log.pdf>. 2013. Acesso em: 09 out. 2024.

CORREIA, M. B. **Algoritmos genéticos**. Dos Algarves, ESGHT, Universidade do Algarve, n. 12, p. 36–43, 2004.

DUTRA, A. L. L. **Otimização do processo logístico de transporte**: estudo de caso em uma indústria de alimentos. Pontifícia Universidade Católica de Goiás, 2023. Disponível em: <https://repositorio.pucgoias.edu.br/jspui/bitstream/123456789/6064/1/Ana%20Luisa%20Luiz%20Dutra_EngProd.pdf>. Acesso em: 23 out. 2024

ENAMI, L. M. et al. **O problema do caixeiro viajante através de algoritmo genético**. 2019. Disponível em: <https://aprepro.org.br/conbrepro/2019/anais/arquivos/09302019_220914_5d92b20230a58.pdf>. Acesso em: 01 out. 2024.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Reading, MA: Addison-Wesley, 1989.

GREFENSTETTE, J. J. **Proceedings of the First International Conference on Genetic Algorithms and their Applications**. [S.l.]: Psychology Press, 2014.

HOLLAND, J. **Adaptation in Natural and Artificial Systems**: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press, 1975. ISBN 9780472084609. Disponível em: <https://books.google.com.br/books?id=YE5RAAAAMAAJ>. Acesso em: 02 jan. 2025.

LARRANAGA, P. et al. **Genetic algorithms for the travelling salesman problem**: A review of representations and operators. Artificial intelligence review, Springer, v. 13, p. 129–170, 1999. Disponível em: <https://link.springer.com/article/10.1023/A:1006529012972>. Acesso em: 03 jan. 2025.

MITCHELL, M. **An introduction to genetic algorithms**. [S.l.]: MIT press, 1998. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=0eznlz0TF-IC&oi=fnd&pg=IA2&dq=An+introduction+to+genetic+algorithms.+&ots=sjmD6X3cMd&sig=R_BBFU9SnFjX1_b2285cAbyHt7s&redir_esc=y#v=onepage&q=An%20introduction%20to%20genetic%20algorithms.&f=false>. Acesso em: 05 jan. 2025.

LANDIM, Anderson. *Software de Otimização de Rotas*. 2025. Disponível em: <Software-de-otimiza-o-de-rotas/Software_de_Otimização_de_rotas.ipynb at Software-de-otimização-de-rotas · AndersonLandim2025/Software-de-otimiza-o-de-rotas>.Acesso em: 3 abr. 2025.