Universidade Estadual de Londrina
Centro de Tecnologia e Urbanismo
Departamento de Engenharia Elétrica

Vinícius Dourado Silva

Otimização da Eficiência de Entregas: Uma Análise Comparativa de Algoritmos para o Problema do Caixeiro Viajante

Londrina
2024

Universidade Estadual de Londrina

Centro de Tecnologia e Urbanismo
Departamento de Engenharia Elétrica

Vinícius Dourado Silva

Otimização da Eficiência de Entregas: Uma Análise Comparativa de Algoritmos para o Problema do Caixeiro Viajante

Trabalho de Conclusão de Curso orientado pelo Prof. Dr. Ernesto F. Ferreyra Ramirez intitulado "Otimização da Eficiência de Entregas: Uma Análise Comparativa de Algoritmos para o Problema do Caixeiro Viajante" e apresentado à Universidade Estadual de Londrina, como parte dos requisitos necessários para a obtenção do Título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Ernesto F. Ferreyra Ramirez

Londrina
2024

Ficha Catalográfica

Vinícius Dourado Silva
Otimização da Eficiência de Entregas: Uma Análise Comparativa de Algoritmos para o Problema do Caixeiro Viajante - Londrina, 2024 - 83 p., 30 cm.
Orientador: Prof. Dr. Ernesto F. Ferreyra Ramirez
1. Programação. 2. Inteligência Artificial. 3. Otimização. 4. Algoritmos.
I. Universidade Estadual de Londrina. Curso de Engenharia Elétrica. II. Otimização da Eficiência de Entregas: Uma Análise Comparativa de Algoritmos para o Problema do Caixeiro Viajante.

Vinícius Dourado Silva

Otimização da Eficiência de Entregas: Uma Análise Comparativa de Algoritmos para o Problema do Caixeiro Viajante

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Estadual de Londrina, como requisito parcial para a obtenção do título de Bacharel em Engenharia Elétrica.

Comissão Examinadora

Prof. Dr. Ernesto F. Ferreyra Ramirez
Universidade Estadual de Londrina
Orientador

Prof. Dr. Francisco Granziera Junior
Universidade Estadual de Londrina

Prof. Dr. Leonimer Flávio de Melo
Universidade Estadual de Londrina

Londrina, 27 de março de 2024

Dedico este trabalho a todos aqueles que, de alguma forma,
auxiliaram para a concretização desta etapa.

Agradecimentos

Adasdasdadasdsadasddasda.

"One day, you'll leave this world behind So live a life you will remember"
(Avicii)

Vinícius Dourado Silva. **Otimização da Eficiência de Entregas: Uma Análise Comparativa de Algoritmos para o Problema do Caixeiro Viajante**. 2024. 83 p. Trabalho de Conclusão de Curso em Engenharia Elétrica - Universidade Estadual de Londrina, Londrina.

Resumo

dasdasdasdasdsadasd

Palavras-Chave: 1. Programação. 2. Inteligência Artificial. 3. Otimização. 4. Algoritmos.

Vinícius Dourado Silva. **There won't be coup**. 2024. 83 p. Monograph in Electrical Engineering - Londrina State University, Londrina.

Abstract

dasdasdasdasdadadasdadadasd

Key-words: 1.Programming. 2. Artificial Intelligence. 3. Optimization. 4. Algorithms.

Lista de ilustrações

Lista de tabelas

Lista de quadros

Lista de Siglas e Abreviaturas

ABRASEL	Associação Brasileira de Bares e Restaurantes
PCV	Problema do Caixeiro Viajante
IA	Inteligência Artificial
AG	Algoritmo Genético
Fig.	Figura
RNA	Redes Neurais Artificias
SA	*Simulated Annealing*
Eq.	Equação
API	Interface de programação de aplicações

Sumário

1 Introdução

1.1 Motivação e Justificativa

1.2 Objetivos

1.2.1 Objetivo Geral

1.2.2 Objetivos Específicos

1.3 Organização do Trabalho

2 Fundamentação Teórica

2.1 Logística na era Pós-Pandemia

Com a chegada da era pós-pandemia, a logística enfrenta uma série de desafios e oportunidades que moldam o futuro do setor. A crise econômica mundial foi a principal responsável por acelerar o uso de ferramentas digitais na logística, visando o benefício e simplificação das relações de comércio, bem como diferentes estratégias de otimização, visando o aprimoramento nos diversos setores de transporte (LMXlogistica, 2022).

Atualmente com a concorrência entre as empresas, se torna necessário estar em constante mudança para se destacar no mercado. Pois, devido à grande velocidade de inovação e surgimento de novas tecnologias, as empresas que não conseguem acompanhar este ritmo tendem a se tornar obsoletas e podem até fechar as portas (HUGO et al., 2017).

Carvalho et al. (1980) argumentam que a gestão de transportes, no que tange à redução de custos, é um impasse para as empresas nos dias de hoje. Ballou (2004) complementa que a logística participa de pelo menos um terço total das despesas de uma empresa. Considerando, o custo e o tempo, é inevitável que o planejamento da distribuição, não apenas de transporte de produtos, mas também de pessoas seja cada vez mais complexo.

Conforme destacado por Wang (2016), a logística acompanhou o desenvolvimento da revolução industrial, alcançando marcos significativos ao longo do tempo. Um desses marcos foi a terceira inovação, que consistiu na sistematização da gestão logística com a introdução de computadores e Tecnologia da Informação (TI). Além disso, a quarta inovação, impulsionada principalmente pela Internet das Coisas (IoT) e Big Data, visa economizar mão de obra e estabelecer padrões no gerenciamento da cadeia de suprimentos.

A implementação destes sistemas logísticos otimizados gera um grande impacto no lucro dos empreendimentos, por meio da diminuição de tempo ou de custos nas atividades rotineiras, como organizar um armazém ou a frota de entregas de uma grande empresa (Mobilidade Sampa, 2023). Temos como exemplo a *YMS Trackage Maestro*, onde os números mostram que o uso do seu próprio software de gestão de pátio, denominado de *YMS Trackage*, onde o sistema de gestão de pátio possibilitou uma redução de até 60% das filas de veículos na portaria, além de um aumento de até 13% na produtividade (DIAS, 2022).

A crise econômica foi fator fundamental para acelerar o uso de ferramentas digitais na logística, com o benefício da simplificação das operações de comércio. Abordando o segmento alimentício, o serviço de entrega a domicílio (*delivery*) é uma prática que está presente no cotidiano de inúmeras pessoas. Segundo a Associação Brasileira de Bares e Restaurantes (ABRASEL), o sistema de *delivery* movimentou cerca de R$11 bilhões em 2019.

Nesse sentido, destaca-se o Problema do Caixeiro Viajante (PCV), que propõe um algoritmo para otimização de rotas e que abrange várias adaptações na atualidade (BARBOSA; JR.; KASHIWABARA, 2015).

A partir daqui, iremos explorar a aplicação de Inteligência Artificial (IA) na resolução de problemas complexos como o PCV (secção 2.3). Inicialmente, vamos nos aprofundar nos Algoritmos Genéticos, uma classe de algoritmos de busca inspirados na teoria da evolução natural (secção 2.3.3). Em seguida, vamos explorar o *Simulated Annealing* (secção 2.4), uma técnica de otimização meta-heurística inspirada no processo de recozimento em metalurgia. Por fim, iremos utilizar a linguagem de programação Python (secção 2.5), amplamente adotada na área de IA, para implementar e testar nossos algoritmos.

2.2 Problema de Otimização do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) teve seus registros iniciais a pouco menos de 100 anos atrás, que foi citado pela primeira vez por Júlia Robinson em um relatório para *RAND Corporation* (ROBINSON, 1949), quando o poder computacional não existia ou era uma fração do que é hoje e continua amplamente abordado no mundo atual.

O PCV é um desafio clássico de otimização cujo objetivo é encontrar a rota mais eficiente que um vendedor (o caixeiro viajante) pode realizar para visitar uma série de cidades apenas uma vez, antes de retornar ao ponto de partida (BENEVIDES et al., 2011). Uma abordagem do PCV estabelece uma única rota que passe por cada nó de um grafo, como visto na Figura 1, apenas uma vez, retornando ao nó inicial no final do percurso. O problema tem implicações em diversas áreas, incluindo logística, transporte e comunicação (CARVALHO, 2018)

Para Souza e Romero (2014), o PCV pode ser retratado como a formação de uma rota que se inicia e termina no mesmo ponto (vértice), após passar por vários locais, a fim de reduzir custos, tempo e extensão da viagem.

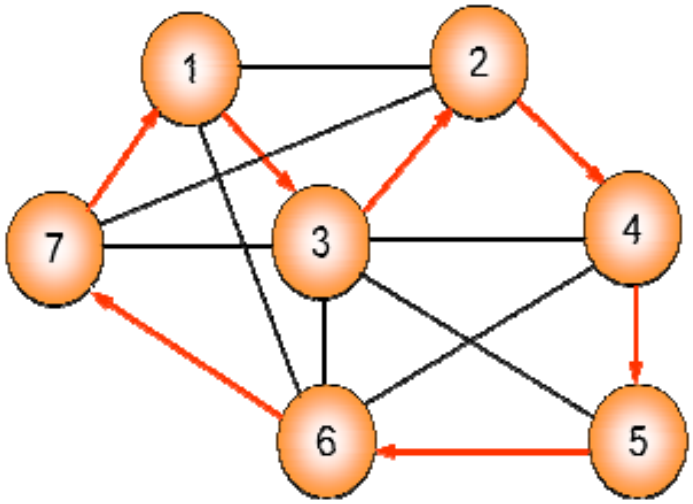


Figura 1 – Exemplo do PCV (BENEVIDES et al., 2011).

À medida que o número de nós (pontos) aumenta, a complexidade da resolução do PCV também aumenta. Para um problema com "n" nós e uma ligação entre cada par de nós, o número de rotas possíveis pode ser calculado pela fórmula:

$$\frac{(n-1)!}{2} \tag{2.1}$$

onde cada nó possui (n – 1) possibilidades de conexão, e a divisão por dois considera que cada rota tem uma rota reversa equivalente com a mesma distância. Dessa forma, um problema com 10 nós resulta em aproximadamente 181 mil soluções, enquanto um com 20 nós pode ter até $6,08 \times 10^{16}$ soluções, e assim sucessivamente (LIEBERMAN, 2010).

De acordo com Prestes (2006), como os tempos para otimizar rotas com vários vértices por algoritmos exatos são inviáveis, a opção nesses casos seria o uso de heurísticas, onde os métodos utilizados são baseados em experiências e regras práticas que encontram soluções mais rapidamente mas não necessariamente exatas. A implementação da Heurística de construção de rotas para o PCV consiste em algoritmos que geram um circuito viável partindo de conjunto inicial de vértices, e modificando esse conjunto a cada iteração utilizando um critério de escolha (SILVA et al., 2013).

Há também diversas abordagens para o problema do caixeiro viajante utilizando Inteligência Artificial (secção 2.3). Essas abordagens se distinguem não apenas em termos de parâmetros utilizados, mas também na maneira como representam soluções possíveis.

2.2.1 API do *Google Maps* para Geração da Matriz de Distâncias

A interface de programação de aplicações, API, do *Google Maps* desempenhou um papel crucial na geração da matriz de distâncias necessária para a resolução do PCV.

Através desta API, foi possível adquirir informações precisas sobre as distâncias entre as cidades selecionadas aleatoriamente para realização dos futuros testes.

Dentre as principais características que levaram a utilização da API, temos:

1. **Acesso a Dados Precisos:** A API fornece dados precisos e atualizados sobre as distâncias entre diferentes localidades, assegurando a confiabilidade das informações utilizadas no processo de otimização do PCV.

2. **Integração com o Algoritmo:** As informações obtidas através da API foram integradas diretamente ao algoritmo de resolução do PCV. Isso permitiu que o sistema calculasse as rotas mais eficientes, levando em consideração as distâncias reais entre as cidades.

3. **Eficiência na Otimização:** Com a matriz de distâncias gerada pela API do *Google Maps*, o algoritmo conseguiu encontrar soluções otimizadas para o PCV. Isso considerou as distâncias reais entre os pontos, garantindo uma roteirização eficiente.

4. **Facilidade de Implementação:** A API do *Google Maps* oferece uma interface amigável e documentação detalhada, facilitando a integração com o sistema desenvolvido para resolver o PCV, mesmo para desenvolvedores com pouca experiência prévia.

2.3 Inteligência Artificial

A Inteligência Artificial (IA) é um campo da ciência da computação que se concentra no desenvolvimento de sistemas e algoritmos capazes de realizar tarefas que normalmente exigiriam inteligência humana (MCCARTHY, 2007). Essas tarefas incluem o reconhecimento de padrões, a tomada de decisões, o aprendizado de novos conceitos e a resolução de problemas complexos. A IA busca replicar o funcionamento do cérebro humano em máquinas, permitindo que elas aprendam com dados, se adaptem a novas situações e executem tarefas de forma autônoma.

Os avanços na IA têm impulsionado diversas áreas, desde a automação industrial até a medicina, passando pela análise de dados, reconhecimento de voz, jogos e entre outros (CENTER", 2023). Com algoritmos e pode computacional cade vez maiores e mais complexos, a IA tem transformando a maneira como interagimos com a tecnologia e como o mundo funciona.

Seja em assistentes virtuais, veículos autônomos, sistemas de recomendação ou diagnósticos médicos, entre outras aplicações, a Inteligência Artificial está presente em nosso cotidiano e seu potencial para revolucionar diversos setores é vasto e promissor (PINHEIRO; OLIVEIRA, 2022).

Um dos exemplos em que mais se faz necessário o uso da IA, é o setor de otimização das entregas, representando uma questão crucial na atualidade para muitas empresas, principalmente naquelas que operam em setores altamente competitivos, como o comércio eletrônico. Para competir com eficácia, os empreendimentos precisam entregar seus produtos rapidamente e com eficiência, justamente onde a otimização de rotas se torna fundamental, especialmente quando focada para pequenas empresas, tornando assim a competição com grandes empresas mais justa (ecommercebrasil, 2023).

Para aplicar esse conceito nas empresas, é necessário uma preparação prévia, pois segundo Ailton Oliveira, cientista de dados da Trackage (DIAS, 2022):

> *"Um dos maiores desafios da inteligência artificial dentro da logística é a disponibilidade de dados. Isso acontece porque muitas vezes as empresas confundem quantidade de dados, ou seja, volume de dados, com dados de qualidade."*

Ailton (DIAS, 2022) complementa estas informações ao explicar que os dados fornecidos precisam ter um significado relevante para o aprendizado da IA e assim melhorar o trabalho realizado.

2.3.1 História

A história da inteligência artificial remete aos anos 50, com os trabalhos pioneiros de Alan Mathison Turing, conhecido popularmente como o pai da computação. Foi o principal responsável pela criação da Máquina de Turing, a qual foi utilizada para quebrar a criptografia da Enigma, dispositivo de criptografia utilizado pela Alemanha durante a Segunda Guerra Mundial para codificar mensagens secretas, causando uma verdadeira guinada na guerra para os aliados, mas a pesquisa em IA teve altos e baixos, com períodos de entusiasmo e desilusão, após essa grande descoberta (Roberto N. Onody, 2021).

Turing também foi além, afim de responder a principal dúvida sobre IAs, *afinal elas podem pensar?* Para aferir isso, Turing propôs o teste de Turing, onde o desempenho de uma máquina perante a tarefa de simular o comportamento humano e assim possa enganar a inteligência humana (SILVA; ARRUDA, 2016).

Posteriormente, surgiram os primeiros estudos sobre redes neurais para construção de redes neurais artificiais foram publicados (EBERHART, 1990), onde as redes neurais artificiais eram estabelecidas, como o perceptron, um modelo fundamental no campo. As redes neurais artificiais foram estabelecidas como uma abordagem promissora para a IA, onde as decisões são baseadas na multiplicação dos pesos dos neurônios, como visto na Figura 2.

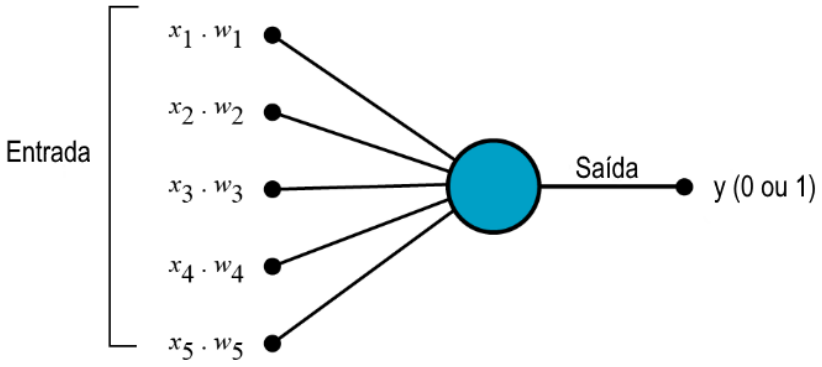


Figura 2 – Exemplo de Perceptron (Deep Learning Book, 2021).

No final da década de 70, a IA passou por um período de estagnação e redução de investimentos, conhecido como "Inverno da IA". Esse período foi caracterizado pelo declínio do interesse e dos recursos financeiros destinados a IA. Diversos fatores contribuíram para essa situação, incluindo expectativas exageradas em relação às capacidades da IA e a ausência de avanços significativos. Além disso, o alto critério dos pesquisadores e as limitações dos dispositivos existentes em termos de poder computacional também foram fatores determinantes (COZMAN, 2018).

Esse conjunção de elementos resultou em uma diminuição tanto do financiamento quanto do interesse acadêmico e industrial na área, evidenciando a necessidade de um realinhamento das expectativas e do desenvolvimento tecnológico para impulsionar o campo da IA.

A segunda era de desenvolvimento da IA, foi marcada pelo surgimento do *perceptron* multicamadas e os algoritmos de retro propagação (*backpropagation*), que permitiram a modelagem de problemas mais complexos e aprimorou a capacidade de aprendizado das redes neurais, ajustando os pesos das conexões entre os neurônios (PINHEIRO et al., 2021).

Esses avanços pavimentaram o caminho para a popularização e aplicação generalizada da IA em uma variedade de campos, contribuindo para o surgimento de tecnologias que utilizamos no dia a dia, como sistemas de recomendação, reconhecimento de voz e visão computacional.

2.3.2 Tipos

A IA pode ser dividida em duas classes, como IA fraca e IA forte ou IA geral. A IA fraca, em inglês *Artificial Narrow Intelligence*, é a teoria de que um sistema de inteligência artificial não teria a capacidade de raciocinar verdadeiramente e resolver os problemas (SALMEN; WACHOWICZ, 2021). De acordo com Quaresma (2021), a IA fraca ainda segue evoluindo com sistemas mais aprimorados e vem atuando nas demais frentes da

sociedade atual, como um aliado do ser humano. Salmen e Wachowicz (2021) ainda citam exemplos de IA fraca, como reconhecimento facial, de lugar, de objetos, de animais ou também na forma de *chatbot* (Software que simula uma conversa humana programado previamente).

A IA forte ou geral, em inglês *Artificial General Intelligence*, demonstra a forma de inteligência baseada em um computador, na qual, é capaz de raciocinar e solucionar qualquer problema (SALMEN; WACHOWICZ, 2021). A IA forte também pode ser definida como a hipótese de que as máquinas podem apresentar consciência, como a dos humanos (CAMPOS, 2021).

Em meados dos anos 80, John Searle afirmou que os computadores seriam autônomos e que teriam capacidade de raciocinar e de terem emoções, ainda, o autor cita que mentes artificiais seria um insucesso, devido a nenhum programa ser o suficiente para fornecer um sistema inteligível (SEARLE, 1987).

Nesse sentido, há os subtipos de IA: Simbólica, RNAs, Lógica *Fuzzy* e Algoritmo Genético. A IA simbólica ou então Sistemas Especialistas, são aqueles que podem resolver problemas a partir de regras já definidas, não generalizando suas respostas. Na implementação desse sistema, um especialista (humano) irá reger o sistemas com as regras necessárias, e posteriormente, o sistema retribuirá com respostas coerentes para certo problema (MARTINS, 2010).

Segundo Martins (2010), na prática do sistema simbólico, normalmente se baseia em um tipo de diálogo entre a máquina e o ser humano e assim, o sistema pode chegar ou não em uma conclusão.

Outro tipo de IA, a Redes Neurais Artificiais ou RNAs, são sistemas que geram classificações de modo automático ou semi automático, elas são configuradas para classificar ou reconhecer informações (MARTINS, 2010). Ela está inserida na área de *machine learning*, é como um modelo matemático que tenta simular a estrutura e funcionalidades de uma rede neural biológica, o uso do RNAs é como uma ferramenta potencial que auxilia na redução dos problemas observados nos modelos tradicionais de gerenciamento de resultados (SILVEIRA et al., 2023).

Em outras palavras Moreira et al. (2021) definem RNAs como sistemas paralelos que são compostos por unidades de processamento simples, que são capazes de calcular funções matemáticas. As unidades de processamento simples estão organizadas em uma ou mais camadas e interligadas por conexões, na maioria, unidirecionais.

No entanto, O pai da lógica *fuzzy*, Lofti Zadeh, divulgou em 1965 o artigo *"Fuzzy Sets"*, onde ele propôs uma teoria de conjuntos onde não existe descontinuidade, ou seja, que não existe uma interrupção abrupta entre os pertencentes e não pertencentes de um conjunto (ZADEH, 1965).

Segundo Zadeh (1965), a lógica *fuzzy* pode ser definido como:

> Com **X** sendo um espaço de objetos, e um elemento qualquer de

> **X**, sendo denotado por **x**. Assim: **X** = **x**. Um conjunto *fuzzy* **A** em **X** é caracterizado por uma função de pertencimento **fA**(**x**), que associa cada ponto em **X** a um número real no intervalo [0, 1], com o valor de **fA**(**x**) em **x** representando o "grau de pertencimento de **x** em **A**. Assim, quanto mais próximo de 1 for o valor de **fA**(**x**), maior o pertencimento de **x** em **A**.

Esse tipo de IA, propõe tratar de problemas dentro da IA, reduzindo as dificuldades em diversos setores, como os problemas de imprecisão nos dados e a incerteza do conhecimento. Em 2004, Russell e Norvig (2004) afirmaram que a lógica *fuzzy* era considerada como uma solução eficiente. Com suas numerosas possibilidades práticas, o sistema *fuzzy* pode ser considerado como um avanço nas metodologias de aplicação em produtos industriais, facilitando os processos matemáticos (VÁZQUEZ-GUZMÁN; MARTÍNEZ-RODRÍGUEZ; SOSA-ZÚÑIGA, 2015). Além disso, o último dos principais tipos de IA, é o Algoritmo Genético (AG), retratado na subseção 2.3.3.

2.3.3 Algoritmos Genéticos

Algoritmos Genéticos (AG), são algoritmos gerais e normalmente são aplicados em problemas complexos, com grandes espaços de busca e difícil modelagem (GOUVEIA et al., 2021). Os AG são considerados como métodos de otimização e se inspira nos mecanismos de evolução de populações de seres vivos (LACERDA; CARVALHO, 1999).

Em meados dos anos 70, os AG foram propostos pelo cientista e professor John Holland e popularizado por seu aluno, David Goldberg. Holland teve como objetivo de modelar uma implementação computacional de seleção baseado na adaptação da natureza, onde foi descrita pela teoria da evolução das espécies de Charles Darwin (ARAÚJO, 2021). Devido a isso, a linguagem do método é ligada à área biológica e possui nomenclaturas como apresenta-se no Quadro 1.

Quadro 1 – Descrição dos termos biológicos relacionados a AG. Adaptado de Pacheco et al. (1999).

Termo biológico	**AG**
Alelo	Valor da característica
Cromossomo	Palavra binária, vetor, matriz, etc
Fenótipo	Estrutura submetida ao problema
Gene	Característica do problema
Genótipo	Estrutura
Geração	Ciclo
Indivíduo	Solução
População	Conjunto de indivíduos
Loco	Posição na palavra, vetor

Segundo Barreto (1999), os AG são métodos tradicionais de busca e otimização que se baseiam em quatro critérios, que são:

> a. Trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros do problema;
> b. Operam em uma população e não em pontos isolados, reduzindo o risco de busca;
> c. Utilizam informações de custo ou recompensa (informações da função objetiva – *payoff*) e não derivadas ou outro conhecimento auxiliar;
> d. Procedem a busca utilizando operadores escolásticos e não determinísticos.

A busca pela melhor solução se dá a partir da população inicial, que quando combinado os melhores desta população, obtém uma nova população, substituindo a inicial. A cada mutação ou então, cruzamento, é formada uma nova população que apresenta novas e melhores soluções para o problema, assim, atingindo os melhores resultados (FILITTO, 2008), conforme é apresentado na Figura 3.

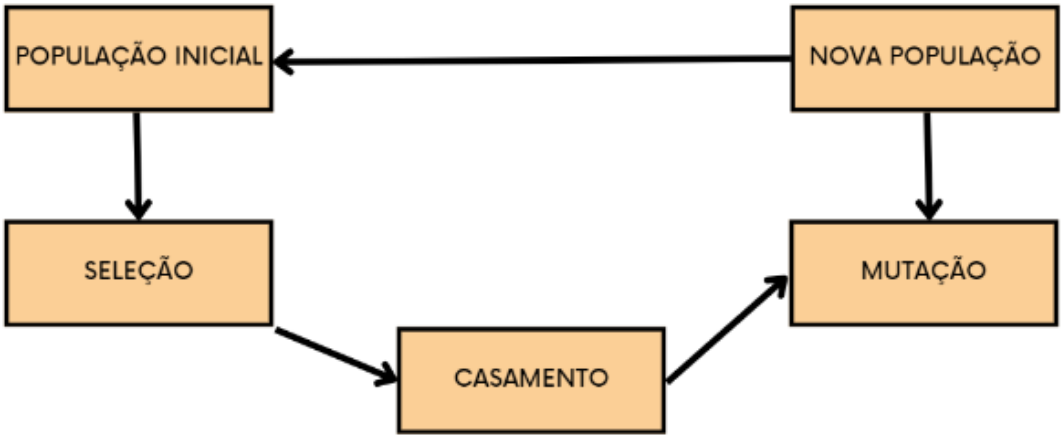


Figura 3 – Fluxograma básico dos algoritmos genéticos.
Fonte: Adaptado de (GOUVEIA et al., 2021)

A principal ideia do processo de seleção é permitir que os indivíduos mais adaptados tenham maior chance de se reproduzir. Barboza e Olandoski (2005) afirmam que a seleção elitista consiste em copiar ou reproduzir os melhores indivíduos da população atual para a próxima geração, garantindo que estes cromossomos não sejam destruídos nas etapas de recombinação e mutação. Sua vantagem é que se no caso ótimo global for descoberto durante o processo de busca, o algoritmo deve convergir para tal solução

Portanto, existe uma tendência de convergência rápida para uma região de mínimos locais ao invés de mínimos globais. Para que isso não ocorra, impõe-se uma rotina para explorar outras áreas do espaço de busca por meio de alterações nos genes por meio da

mutação. A probabilidade de se efetuar uma mutação deve ser relativamente baixa, caso contrário o algoritmo se comportará fazendo uma busca aleatória

2.4 Simulated Annealing

Outra abordagem para solução do PVC se baseia da técnica de otimização conhecida por *Simulated Annealing* (SA), técnica que foi usada para simular em um computador o processo de *"annealing"* de cristais, processo este de resfriamento gradativo de materiais a partir de uma alta temperatura levando-os as estados mínimos de energia.

A ideia de aplicar este método para resolver problemas de otimização combinatória foi proposta por Kirkpatrick et al. (1983).

No recozimento, os metais são aquecidos a altas temperaturas e depois resfriados lentamente, permitindo que os átomos se reorganizem em uma estrutura mais estável. Da mesma forma, o SA começa com uma solução inicial e, em seguida, perturba essa solução de maneira controlada para explorar o espaço de busca (KIRKPATRICK et al., 1983).

O fato do método *Simulated Annealing* permitir a aceitação de configurações intermediárias do problema em que cresce o valor da função objetivo que se deseja minimizar é crucial. Essa aceitação temporária de soluções "piores", significa que o método admite caminhar "morro acima", na esperança de encontrar "vales" mais profundos (BENEVIDES et al., 2011).

O SA é especialmente útil para problemas de otimização complexos, onde métodos de busca local podem ficar presos em mínimos locais. Ao permitir movimentos que aumentam a função objetivo (ou seja, pioram a solução), o SA pode escapar desses mínimos locais e explorar mais completamente o espaço de busca. A probabilidade de aceitar uma solução pior diminui gradualmente à medida que o "resfriamento" do algoritmo prossegue, de forma análoga ao processo de recozimento (CRISTINA et al., 2023).

O SA tem sido aplicado com sucesso a uma variedade de problemas complexos. Por exemplo, tem sido utilizado na criptografia e no projeto de quantizadores vetoriais. Dentre as aplicações práticas mais conhecidas do Problema do Caixeiro Viajante, destacam-se a sequência das operações de máquinas em manufatura, otimização de perfurações de furos em placas de circuitos impressos e a maioria dos problemas de roteamento de veículos (NASCIMENTO et al., 2004).

Podendo convergir para aplicações mais específica, como a otimização rotas de entregas de materiais em uma rede hospitalar (FONSECA et al., 2020), otimização de rotas em uma concessionária de energia elétrica (FERREIRA, 2020), até o planejador de roteiros turísticos (BISPO, 2018).

No entanto, apesar de suas vantagens, o SA também tem suas desvantagens. Por ser um método estocástico, o SA pode ser mais lento do que outros métodos de otimização. Além disso, o desempenho do SA pode ser sensível à escolha dos parâmetros, como a

temperatura inicial e a taxa de resfriamento. Na Figura 4 demonstra um fluxograma genérico do SA.

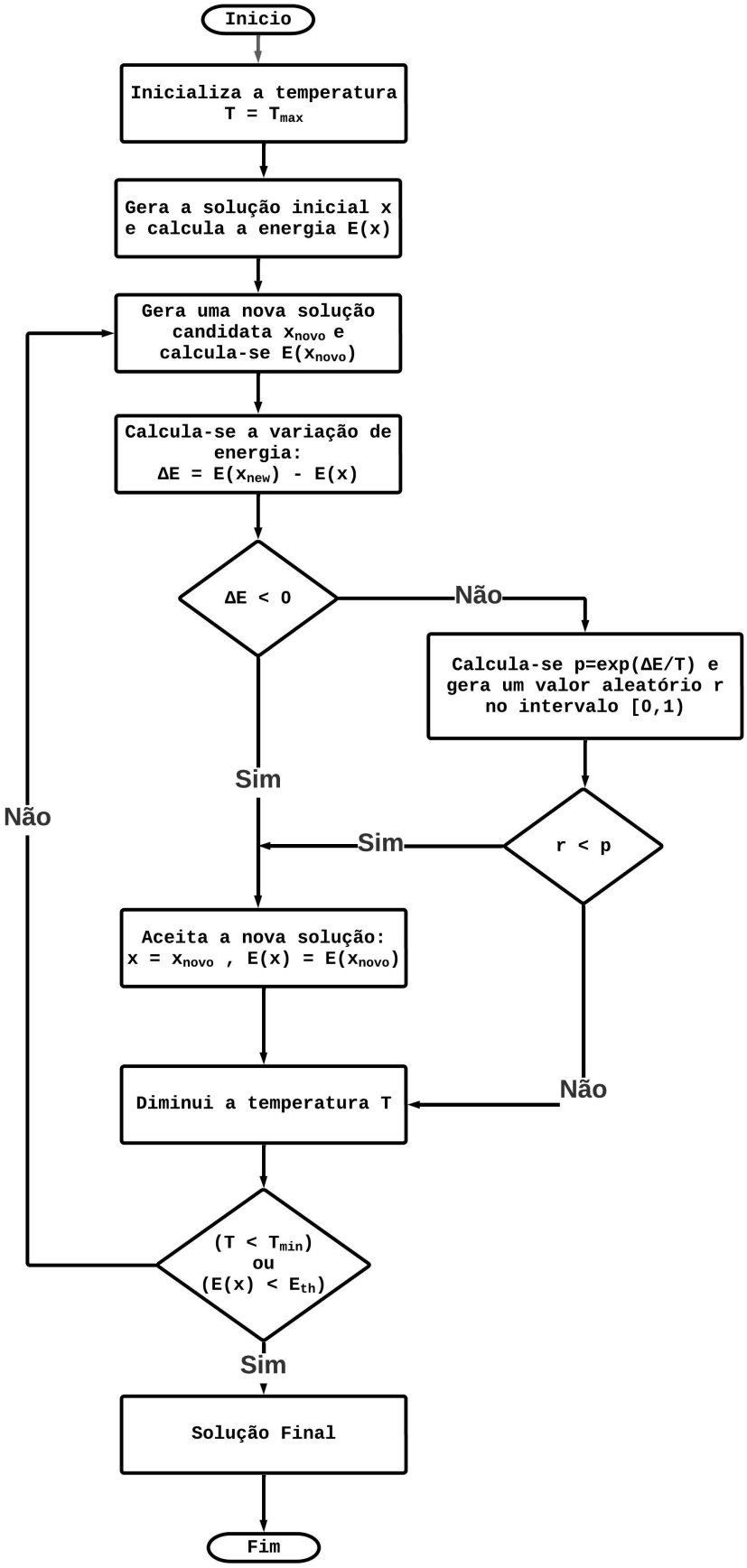


Figura 4 – Exemplo genérico Simulated Annealing.
Fonte: O Autor.

2.5 Python

O *Python* é uma linguagem de programação de alto nível e de propósito geral. Criada por Guido van Rossum em 1982 (SILVA; SILVA, 2019), para evitar as complexidades e os problemas da linguagem C. Lançada pela primeira vez em 1991. Desde então, tornou-se uma das linguagens de programação mais populares do mundo devido à sua simplicidade, legibilidade e vasta gama de aplicações.

Como os principais destaques do *Python*, temos sua sintaxe limpa e facilmente legível, tornando o código mais amigável, facilitando suas manutenções futura. Isso faz com que seja uma ótima escolha para iniciantes em programação, mas também é utilizada por desenvolvedores experientes em várias áreas, como desenvolvimento web, ciência de dados, automação, inteligência artificial, entre outros.

Além da legibilidade, *Python* é conhecido por sua extensa biblioteca padrão, que oferece uma vasta gama de módulos e ferramentas para facilitar o desenvolvimento de aplicativos em diversas áreas. Esses módulos cobrem desde manipulação de arquivos até desenvolvimento de interfaces gráficas de usuário.

Outro aspecto importante do *Python* é sua comunidade ativa e engajada. Existem milhares de pacotes de terceiros disponíveis por meio do PyPI (*Python Package Index*), o que permite aos desenvolvedores acessar uma ampla gama de funcionalidades adicionais para seus projetos, além de redes sociais focadas em desenvolvedores, para assistência e aprimoramento dos códigos.

2.5.1 Interface gráfica em Python

O *Python* oferece diversas bibliotecas para criação de interfaces gráficas, como *Tkinter*, PyQt, wxPython e Kivy. Cada uma dessas bibliotecas tem suas próprias características e vantagens, sendo escolhida de acordo com as necessidades específicas do projeto.

Sendo uma das mais populares a Tkinter, que é uma biblioteca nativa do *Python* e proporciona uma maneira simples de construir interfaces gráficas para aplicativos *desktop*.

A Tkinter permite a criação de janelas, botões, caixas de texto, menus e outros componentes de interface de forma intuitiva e eficiente, Figura **??**. Com ela, é possível desenvolver aplicações com uma variedade de recursos visuais, tornando a interação do usuário mais amigável e produtiva.

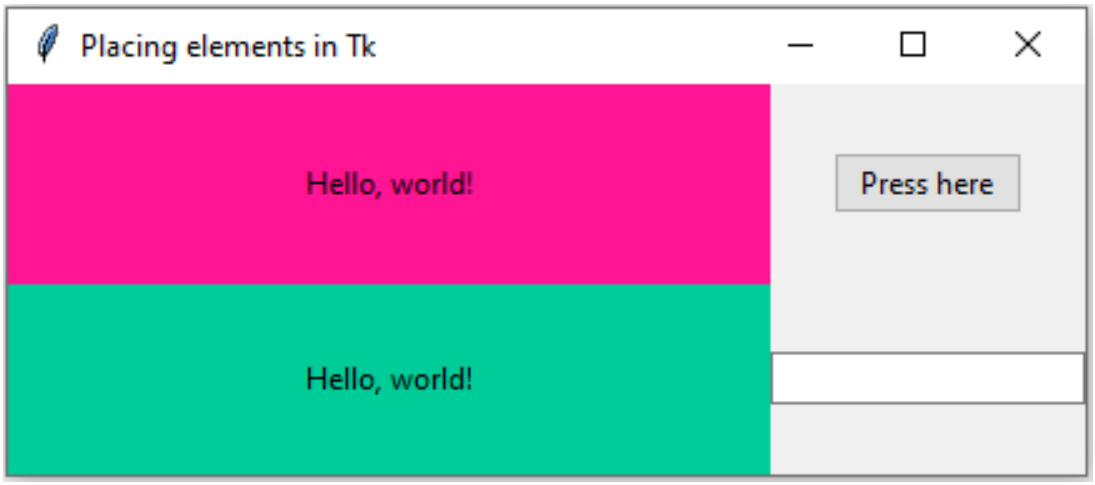


Figura 5 – Exemplo genérico Tkinter.
Fonte: O Autor.

Com a crescente popularidade de *Python* e sua vasta gama de bibliotecas para desenvolvimento de interfaces gráficas, criar aplicativos *desktop* com uma interface visual atraente e funcional tornou-se mais acessível e eficiente para desenvolvedores de todos os níveis de experiência.

3 Desenvolvimento

3.1 Utilização dos Algoritmos

Para estabelecer um linha de otimização para o PCV, três métodos foram escolhidos, sendo eles, o método da Busca Exaustiva, Algoritmo Genético de seleção simples e *Simulated Annealing*. Cada método será detalhado em termos de sua implementação, justificativa para sua escolha e características específicas.

Inicialmente, todos os métodos foram feitos em um ambiente de testes proporcionado pelo software *Matlab*, da *MathWorks*, sendo este um programa com alta capacidade para resolução de cálculos matemáticos, visualização de resultados, além de uma variedade de *toolbox* e documentação de fácil acesso na internet.

Todos os códigos abaixo foram executados em uma máquina com as seguinte configurações:

- **Processador:** Processador Intel(R) Core(TM) i3-4330 CPU @ 3.50GHz 3.50 GHz
- **Memória RAM:** 16,0 GB
- **Tipo de Sistema:** *Windows* 10 Pro, Sistema operacional de 64 bits, processador baseado em x64

3.1.1 Busca exaustiva

A busca exaustiva é um método direto e sistemático de encontrar a solução ótima para um problema, considerando todas as combinações possíveis de soluções. Portanto, o código necessário deve ser o mais simples possível. Neste trabalho, a busca exaustiva foi escolhida devido à sua garantia de encontrar a solução ótima em problemas com espaços de busca pequenos e pela sua simplicidade de implementação (BAIDOO; OPPONG, 2016).

O algoritmo de busca exaustiva foi implementado de acordo com os seguintes passos:

1. Geração de todas as possíveis soluções para o problema.
2. Avaliação de cada solução gerada.
3. Seleção da solução ótima com base nos critérios de avaliação definidos.

Seguindo o seguinte fluxograma 6, que foi implementado primeiramente em *Matlab*, apêndice 5.1 depois em *Python*, apêndice 5.1:

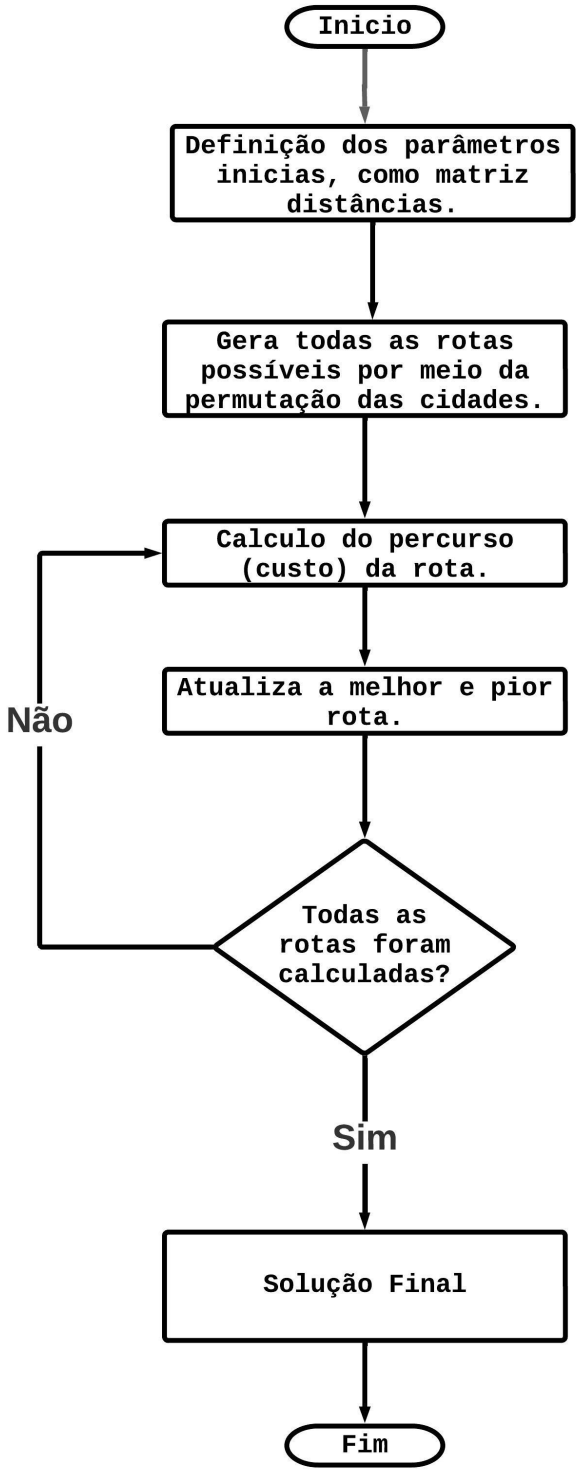


Figura 6 – Fluxograma da busca exaustiva.
Fonte: De autoria própria.

Nesse sentido, a metodologia proposta envolve a utilização de uma matriz de distâncias das cidades selecionadas para os testes, além das permutações das cidades, que representam as diferentes rotas a serem consideradas. Com essas informações disponíveis, é possível calcular o custo de cada rota e, em seguida, identificar a rota de menor e maior

custo, bem como determinar o tempo necessário para a realização da simulação.

Apesar da sua garantia de encontrar a solução ótima, a busca exaustiva pode se tornar computacionalmente inviável para problemas com espaços de busca muito grandes, devido ao seu alto custo computacional, no caso do PCV, seguindo a equação 2.1.

3.1.2 Algoritmo Genético de seleção simples

O algoritmo genético proposto busca solucionar o Problema do Caixeiro Viajante aplicando a técnica de otimização baseada em processos evolutivos. Este método foi escolhido devido à sua capacidade de lidar com espaços de busca grandes e complexos, além de sua flexibilidade e adaptabilidade.

A implementação do algoritmo genético incluiu os seguintes passos:

1. Geração inicial de uma população de soluções.

2. Avaliação de cada indivíduo da população gerada anteriormente.

3. Seleção dos indivíduos mais aptos para reprodução.

4. Aplicação de operadores genéticos (*crossover* e mutação) para criar uma nova população.

5. Avaliação da nova população e repetição do processo até atingir um critério de parada.

Seguindo o seguinte fluxograma 7, que foi implementado primeiramente em *Matlab*, anexo 5.1 depois em *Python*, anexo 5.1:

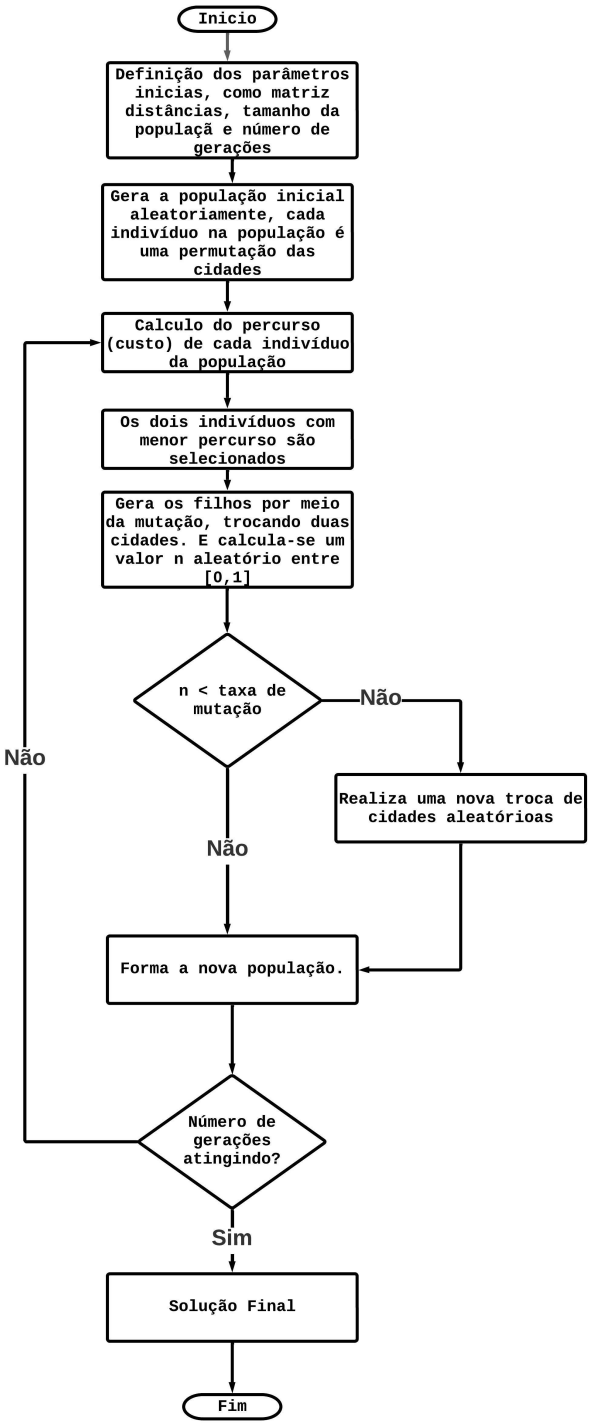


Figura 7 – Fluxograma do algoritmo genético de seleção simples.
Fonte: De autoria própria.

Na etapa de inicialização, é necessário definir as cidades a serem testadas, bem como os parâmetros necessários. Isso inclui determinar o tamanho da população inicial e o número máximo de gerações. Para cada cromossomo na população inicial, um vetor de "n" posições, ou seja, "n" cidades, é gerado, com os índices das cidades distribuídos aleatoriamente.

Os parâmetros do algoritmo genético foram ajustados empiricamente para obter os melhores resultados, levando em consideração a taxa de *crossover*, a taxa de mutação e o tamanho da população.

Em seguida, para o processo de evolução, em cada população, o cálculo do percurso total é realizado. Os dois melhores indivíduos são selecionados como pais da próxima população. Cada pai gera filhos através da permutação de duas posições.

Após ter os filhos gerados, o algoritmo prepara para fazer ou não a mutação, tendo como base a taxa definida anteriormente. Posteriormente, ocorre a atualização da população.

Como resultado, o código proposto foi desenvolvido para fins comparativos com os demais métodos de solução. Ele produz observações semelhantes, como o tempo de execução e o menor percurso encontrado ao longo das gerações.

3.1.3 Simulated Annealing

O *simulated annealing* é uma técnica de otimização probabilística baseada no processo de recozimento de metais. Ele foi escolhido por sua capacidade de escapar de mínimos locais e explorar mais completamente o espaço de busca. Além disso, o SA é especialmente útil para problemas com espaços de busca complexos e mal comportados.

A implementação do SA envolveu os seguintes passos:

1. Definição aleatória de uma solução inicial.

2. Geração de uma solução vizinha através de uma perturbação controlada.

3. Avaliação da solução vizinha e decisão de aceitar ou rejeitar com base em uma função de probabilidade.

4. Repetição do processo até atingir um critério de parada.

Seguindo o seguinte fluxograma 8, que foi implementado primeiramente em *Matlab*, anexo 5.1 depois em *Python*, anexo 5.1:

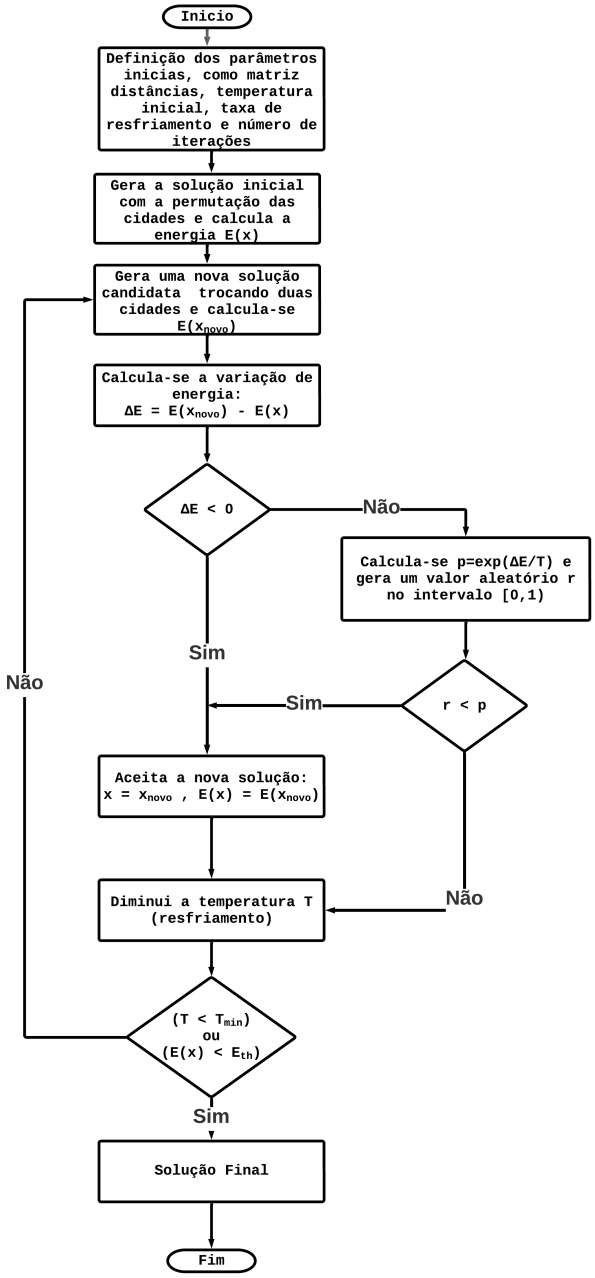


Figura 8 – Fluxograma do *Simulated Annealing*.
Fonte: De autoria própria.

Os parâmetros do *simulated annealing*, como a temperatura inicial (75°C) e a taxa de resfriamento, foram ajustados para equilibrar a exploração do espaço de busca com a intensificação da busca pela solução ótima.

Afim de padronizar os resultados obtidos, o código proposto foi desenvolvido para fins comparativos com os demais métodos de solução. Ele produz observações semelhantes, como o tempo de execução e o menor percurso encontrado ao longo das gerações. Facilitando assim a comparação direta entre todos os códigos propostos.

3.1.4 Banco de Dados

O banco de dados foi desenvolvido contendo informações detalhadas sobre 50 cidades selecionadas aleatoriamente no estado do Paraná, as cidades podém ser vistas no seguinte quadro, quadro 2:

Quadro 2 – Cidades do Paraná escolhidas aleatoriamente para testes.

UEL,Londrina	Lobato, PR
Ibipora, PR	Araruna, PR
Maringa, PR	Jesuitas, PR
Cascavel, PR	Santa Izabel do Oeste, PR
Paranagua, PR	Centenario do Sul, PR
Perola, PR	Sao Joao, PR
Palmas, PR	Florestopolis, PR
Brasilandia do Sul, PR	Contenda, PR
Senges, PR	Guaratuba, PR
Santo Antonio do Sudoeste, PR	Luiziana, PR
Curitiba, PR	Cafeara, PR
Nova Londrina, PR	Castro, PR
Umuarama, PR	Lapa, PR
Arapongas, PR	Toledo, PR
Alvorada do Sul, PR	Piraquara, PR
Roncador, PR	Jacarezinho, PR
Fenix, PR	Matinhos, PR
Maua da Serra, PR	Assai, PR
Cruzeiro do Oeste, PR	Paranacity, PR
Urai, PR	Primeiro de Maio, PR
Cornelio Procopio, PR	Foz do Iguacu, PR
Sapopema, PR	Borrazopolis, PR
General Carneiro, PR	Guapirama, PR
Sao Jose dos Pinhais, PR	Inaja, PR
Indianopolis, PR	Nova Fatima, PR

A seleção aleatória foi empregada com a finalidade de assegurar uma diversidade de cenários para a análise comparativa do desempenho dos algoritmos de resolução do PCV.

A inclusão de 50 cidades no banco de dados possibilita a formação de um conjunto de dados robusto e diversificado, que compreende desde pequenos municípios até grandes metrópoles. Isso permite a análise de uma variedade de rotas, desde as que percorrem longas distâncias entre cidades afastadas até as que se concentram em áreas urbanas com uma malha rodoviária mais extensa.

Cada cidade no banco de dados é caracterizada por uma série de atributos, incluindo seu nome e a distância para todas as outras cidades, além de sua distribuição geográfica. Esses dados são fundamentais para a modelagem e implementação dos algoritmos de resolução do PCV, pois facilitam a criação de rotas realistas e pertinentes para a análise.

A diversidade de cidades e características presentes no banco de dados, gera uma ampla gama de cenários de teste para avaliar o desempenho dos algoritmos em diferentes contextos. Por exemplo, é possível comparar o desempenho dos algoritmos em rotas mais extensas, que englobam várias cidades e regiões geográficas, com o desempenho em rotas mais curtas, que se concentram em áreas urbanas específicas.

3.2 Avaliação do Desempenho Computacional

Primeiramente, para ambos os casos foram escolhidas cidades testes arbitrariamente, bem como o tamanho inicial da população e o numero de gerações, sendo 10 e 100 respectivamente, para o algoritmo genético, e o número de iterações, temperatura inicial e taxa de resfriamento, sendo 100, 0,7 e 0,95 respectivamente, para o *Simulated Annealing*. Além de tirar uma média das 10 soluções de cada método para que se tenha um padrão comparativo em cada resultado.

Para cada método, foi avaliado seu tempo de execução, suas necessidades computacionais, bem como as melhores distâncias encontradas.

Sendo estes critérios os principais responsáveis por orientar os próximos passos para evolução dos algoritmos, apontando onde estão os possíveis problemas, e quais são as áreas de melhoria.

Inicialmente, os algoritmos precisam rodar com eficiência, portanto as métricas de evolução para o desempenho computacional são fundamentais.

3.3 Aplicando refinamentos nos modelos

Com os algoritmos implementados de maneira básica, se fez necessário aplicar refinamentos em todos os métodos, visando aprimorar a eficiência e a precisão dos algoritmos, garantindo resultados mais eficazes e otimizados. Dentre os principais refinamentos temos:

- Busca Exaustiva: Foram exploradas técnicas de otimização e heurísticas para reduzir o tempo de execução e melhorar a escalabilidade da busca exaustiva. Isso pode incluir a implementação de estratégias de poda de busca, como a eliminação de soluções dominadas, e a paralelização do processo de busca para explorar múltiplas soluções simultaneamente.

- Algoritmo Genético: Realizaram-se ajustes nos parâmetros do algoritmo genético, como o tamanho da população, taxa de mutação e critérios de seleção, visando encontrar um equilíbrio ideal entre diversidade genética e convergência para soluções ótimas.

- *Simulated Annealing*: Foram investigadas técnicas para melhorar a eficácia do SA na busca por soluções ótimas. Isso envolveu a exploração de diferentes funções de temperatura e esquemas de resfriamento, bem como a implementação de estratégias adaptativas para ajustar dinamicamente os parâmetros do algoritmo durante a execução.

Ao aplicar esses refinamentos em cada método, espera-se alcançar um desempenho aprimorado e resultados mais precisos na resolução do PCV. O objetivo é desenvolver algoritmos mais eficientes e robustos, capazes de lidar com instâncias do problema de diferentes tamanhos e complexidades, proporcionando soluções de alta qualidade em tempo hábil.

3.3.1 Entrevista para o refinar seguindo parâmetros realistas.

A metodologia adotada para a condução da pesquisa, teve como objetivo analisar o funcionamento das entregas de um estabelecimento. Para alcançar esse objetivo, foi utilizado a abordagem de entrevista estruturada. A entrevista estruturada é uma técnica de pesquisa que envolve a formulação de um conjunto de perguntas predefinidas e a aplicação sistemática dessas perguntas a todos os participantes da pesquisa. Ela permite a coleta de informações de maneira padronizada e consistente, tornando os dados mais comparáveis e confiáveis.

3.3.1.1 Seleção do Estabelecimento

A escolha do estabelecimento para a realização das entrevistas estruturadas foi feita com base em uma recomendação do orientador. O estabelecimento selecionado possui características que o tornam um objeto de estudo relevante para a pesquisa sobre o tema. Além disso, a escolha foi influenciada pela disponibilidade e acessibilidade do estabelecimento, o que facilitou a realização das entrevistas. Além de uma segunda entrevista com um entregador, que trabalha com entregas fixas e *ifood* em um outro estabelecimento, afim de obter comparações e semelhanças do tema em dois lugares distintos, além de comparar o lado do dono do restaurante e do entregador.

A colaboração do estabelecimento foi essencial para o sucesso da pesquisa e permitiu o acesso às informações necessárias para a análise do processo de entregas com poucos pontos. As entrevistas estruturadas foram conduzidas de acordo com um roteiro predefinido, e os dados coletados desempenharam um papel fundamental na compreensão do funcionamento do processo de entregas no estabelecimento.

Seguem as perguntas feitas para ambos os entrevistados:

1. Como funcionam as entregas? Os entregadores são fixos? Funcionam por Ifood, site, whatsapp?

2. Se possuem diferentes meios de fazer o pedido, poderia me dizer uma média de quantos porcento em cada meio
3. Como funciona o repasse dos pedidos? Umas pessoa anota e confere para passar corretamente ao motoboy?
4. Quantas entregas por saída faz o entregador?
5. Como eles priorizam? Tem uma pessoal que calcula a melhor rota?
6. Os entregadores ganham por entrega? Eles ficam responsáveis por organizar a mochila na hora de sair pra entrega?

3.4 Desenvolvimento do Software em Python

O desenvolvimento do software seguiu uma linha do tempo estratégica, focada na implementação e refinamento progressivo dos algoritmos para resolver o Problema do Caixeiro Viajante (PCV).

- Busca Exaustiva: Inicialmente, a ênfase foi a implementação da busca exaustiva, que serviu como ponto de partida fundamental para entender a complexidade do problema e estabelecer uma base sólida para comparação com outros métodos. Durante esta fase, foram exploradas técnicas para otimizar o desempenho da busca exaustiva, visando reduzir o tempo de execução e melhorar a eficiência na busca por soluções ótimas.
- Algoritmo Genético: Após a implementação e análise da busca exaustiva, o foco foi direcionado para o desenvolvimento do algoritmo genético. Esta etapa envolveu a definição de parâmetros-chave, como tamanho da população, taxa de mutação e critérios de seleção, além da implementação das operações genéticas necessárias, como seleção, *crossover* e mutação. Durante o desenvolvimento do algoritmo genético, foram realizados testes extensivos para avaliar sua eficácia e compará-lo com a busca exaustiva.
- Simulated Annealing: Por fim, foi implementado o SA como uma alternativa heurística para resolver o PCV. Esta etapa envolveu a definição de uma função de energia e a configuração dos parâmetros do algoritmo, como temperatura inicial, taxa de resfriamento e número de iterações. O SA foi refinado e ajustado iterativamente com base nos resultados obtidos em experimentos práticos, visando encontrar um equilíbrio entre exploração e exploração do espaço de soluções.

Ao seguir essa linha do tempo, foi possível desenvolver e aprimorar progressivamente os algoritmos para resolver o PCV, garantindo uma abordagem sistemática e eficaz no

desenvolvimento do software. Cada etapa foi cuidadosamente planejada e executada, levando em consideração as lições aprendidas e os *insights* ganhos ao longo do processo de desenvolvimento.

Após o desenvolvimento do código completo, foi decidido migrar os algoritmos para a linguagem *Python*. Essa escolha foi feita com o objetivo de facilitar a implementação de otimizações relacionadas ao tempo de execução dos códigos, aproveitando as vantagens de desempenho oferecidas pela linguagem *Python* e suas bibliotecas otimizadas. Além disso, a mudança para *Python* permitiu uma transição suave entre o desenvolvimento do funcionamento estrutural e da interface gráfica do software.

3.4.1 Funcionamento Estrutural (back-end)

Dentro da implementação do *back-end*, dividiu-se os algoritmos em arquivos separados, onde parte dos códigos implementados em *Matlab* foi facilmente convertido para *Python*. Assim, as cidades não precisam ser definidas arbitrariamente, e sim são oriundos de uma matriz distâncias fornecidas pelo usuário.

Para garantir a eficiência e a modularidade do nosso sistema, cada algoritmo é implementado em um arquivo separado e projetado para ser facilmente integrado ao restante do código. Isso permite comparar e avaliar diferentes abordagens de forma flexível e eficaz.

Em um arquivo principal, que pode ser visto no Apêndice **??**, é responsável por executar os três algoritmos simultaneamente, e portanto tratar os resultados numéricos e gerar gráficos, mantendo assim um padrão de resultados.

Todos estes arquivos são chamadas por meio de funções, mantendo a otimização do algoritmo, onde o responsável por chamar essas funções é a interface gráfica, que será abordada na próxima sessão.

3.4.2 Interface Gráfica (Front-end)

A implementação da interface gráfica foi feita utilizando a biblioteca *Tkinter*, que é responsável por criar uma interface gráfica definida por uma matriz, facilitando assim alocação dos elementos necessários.

Visando um design interativo , a interface segue um padrão limpo e intuitivo. Na primeira coluna o usuário entra com os parâmetros fundamentais necessários: número de cidades, o arquivo .txt, que contém a matriz de distâncias. Ainda na primeira coluna, há um *checkbox* para a execução de testes onde o número de cidades ou pontos aumenta progressivamente, e *checkboxes* de quais métodos executar.

Na coluna central há um campo para resultados do tempo de execução de cada método, os campos de entrada para os parâmetros de testes com incremento do número de cidades e as entradas dos parâmetros específicos de cada método. A última coluna é responsável por exibir os principais resultados, como a melhor rota de cada algoritmo bem como a

distância a ser percorrido, e no canto inferior direito, há um *timer* para acompanhar o tempo de execução total do programa.

4 Resultados

4.1 Resultados(trocar?) da avaliação Computacional

Os primeiros testes foram realizados no *Matlab*, com os algoritmos implementados de forma simplificada, sem quaisquer mecanismos de otimização ou segurança. Os resultados do tempo de execução foram obtidos realizando a média entre 10 simulações, conforme demonstrado na Tabela 1:

Tabela 1 – Comparação do tempo de execução, em segundos, dos algoritmos em diferentes cenários no *Matlab*.

	Busca Exaustiva**	Algoritmo Genético**	*Simulated Annealing***
5 Cidades	0,0080724 s	0,55935 s	0,0044 s
10 Cidades	0,192 s	0,814 s	0,004 s
15 Cidades	*	0,5451 s	0,099822 s

Legenda:

- *: Não foi possível executar devido a grandes requisitos de memória RAM.
- **: Resultados podem variar com o estado da máquina a ser executado.

Ao analisar os tempos de execução dos algoritmos, observa-se que o *Simulated Annealing*, em *Matlab*, apresenta uma execução significativamente mais rápida.

Para analisar as distâncias, ou seja, o objetivo final dos algoritmos, foi executado um teste retornando os valores de 10 simulações, sendo cada simulação com 10 cidades escolhidas aleatoriamente a cada simulação, como apresentado na Tabela 2

Tabela 2 – Comparação das distâncias (em km) obtidas dos algoritmos em 10 cenários no *Matlab*.

Busca Exaustiva	**Algoritmo Genético de seleção Simples**	***Simulated Annealing***
1.825,92 km	1.921,52 km	1.830,92 km
1.587,572 km	1.587,57 km	1.587,57 km
1.173,14 km	1.225,82 km	1.173,14 km
1.076,72 km	1.141,13 km	1.080,99 km
1.299,30 km	1.304,53 km	1.299,30 km
1.810,45 km	2045,64 km	1.811,19 km
915,01 km	944,55 km	915,01 km
1.180,09 km	1.180,09 km	1.180,09 km
1.345,37 km	1.347,41 km	1.345,37 km
1.376,44 km	1.400,27 km	1.400,27 km
4.434,00 km	4.891,00 km	4.434,00 km

Fonte: De autoria Própria

Pode-se notar na Tabela 2, que é possível comparar as menores distâncias exatas do método de busca exaustiva com os demais métodos computacionais. Para o AG, foi possível atingir o valor esperado, ou seja, a menor distância percorrida, em apenas dois casos, enquanto para o SA, o resultado desejado foi atingindo em sete casos. Com isso, sem melhorias, apenas com os códigos implementados de maneira rudimentar, pode-se considerar o SA melhor que o AG.

Migrando os algoritmos para o *Python*, para facilitar melhorias futuras, os resultados podem ser gerados novamente para se obter as comparações necessárias, e justificar se a mudança de linguagem acrescenta algum tipo de melhoria significativa no resultado final ou no tempo de execução.

Na Tabela 3 é apresentado o tempo de execução em *Python*:

Tabela 3 – Comparação do tempo de execução, em segundos, dos algoritmos em diferentes cenários no *Python*.

	Busca Exaustiva**	Algoritmo Genético**	*Simulated Annealing***
5 Cidades	0,00064 s	0,04536 s	0,0087 s
10 Cidades	4,303 s	0,0192 s	0,00724 s
15 Cidades	*	0,0192 s	0,00724 s

Legenda:

- *: Não foi possível executar devido a grandes requisitos de memória RAM.
- **: Resultados podem variar com o estado da máquina a ser executado.

Como pode ser observado, em *Python* as variáveis são pré alocadas forçadamente, gerando uma melhoria significativa no tempo de execução dos algoritmos AG e SA, en-

tretanto, para o método Busca Exaustiva ocorre o inverso, o tempo de execução acaba aumento exponencialmente.

4.2 Resultados(trocar?) após os ajustes computacionais

4.2.1 Ajustes nos códigos.

Para o método de Busca Exaustiva, por ser um método inviável a para um grande número de cidades, o custo para implementar melhorias como paralelização das soluções ou métodos heurísticos, se torna complexo e pouco lucrativo, portanto melhorias como, inicializar as variáveis antes, permutar apenas os números que não são fixos, ou seja, os pontos a serem visitados e não os pontos de partida e chegada, já causam um ganho no tempo de execução.

No AG, foram ajustados os parâmetros para encontrar a melhor sintonia, as variáveis são criadas no começo do algoritmo, economizando memorias e otimizando tempo, e assim como na Busca Exaustiva, permutar apenas as cidades não fixas também gera uma otimização no tempo de execução.

Já para o SA, melhorar a eficácia se mostrou mais difícil, tendo em vista que diferentes funções de temperatura não mostravam um grande ganho pelo custo de implementação, portanto foi optado apenas por encontrar uma melhor sintonia nos parâmetros.

Após as alterações, rodando os testes novamente, porém agora,

SA 1000 0.84 0.82

Quadro 3 – Respostas da Entrevista Estruturada

Pergunta	**Felipe (Dono do Restaurante)**	**Everson (Entregador)**
Pergunta 1	As entregas funcionam principalmente pelo iFood (80%), seguido de aplicativo próprio e site (15%), e uma parcela menor via telefone e WhatsApp (5%). Todos os entregadores são fixos, totalizando 7 profissionais, dos quais 6 estão na rua e 1 é responsável por organizar as entregas e fazer as rotas.	As entregas dependem do modo de trabalho do entregador. Alguns entregadores, como eu, fazem todas as entregas de todos os pedidos, independentemente de serem do iFood, telefone ou WhatsApp. Recebemos um valor fixo pelo trabalho e todas as taxas são de nossa responsabilidade, conforme acordado com o restaurante.
Pergunta 2	Os pedidos por iFood representam cerca de 80%, enquanto pedidos via aplicativo/site próprio são aproximadamente 15%, e pedidos via telefone/WhatsApp compreendem cerca de 5% do total.	A quantidade de entregas varia significativamente e é bem distribuída. Em termos percentuais, aproximadamente 33% dos pedidos são feitos por telefone, WhatsApp e iFood, respectivamente.
Pergunta 3	Todos os pedidos são integrados ao sistema. A equipe confere os pedidos no final do dia para garantir que todas as entregas tenham sido feitas corretamente. Qualquer discrepância é abordada com os entregadores.	Os pedidos são impressos, preparados na cozinha e ficam prontos no balcão. Nossa responsabilidade é conferir se há troco e refrigerantes, enquanto a responsabilidade pelo produto em si recai sobre a cozinha ou o responsável pelo empacotamento.
Pergunta 4	O número de entregas por saída varia de acordo com a demanda e as rotas, podendo variar de 1 a 6 entregas por saída.	O número de entregas por saída é variável, dependendo da rota e da capacidade da caixa. Pode variar de uma única entrega a várias, com situações em que algumas entregas de uma mesma rota ficam a cargo de outro entregador por falta de espaço na caixa.
Pergunta 5	A priorização das entregas é determinada principalmente pelos entregadores, que conhecem bem a região. Não há um aplicativo específico para calcular a rota, mas o estabelecimento oferece orientação sobre a sequência de entregas.	A priorização das entregas não é feita por um aplicativo, mas sim pelos entregadores. Normalmente, pegamos o pedido que está atrasado e ajustamos a rota de acordo com o destino do pedido. Caso não haja outras entregas na mesma di-

4.2.2 Ajustes Pós-Entrevista

Tabela 4 – Respostas da Entrevista Estruturada

Pergunta	**Felipe (Dono do Restaurante)**	**Everson (Entregador)**
Pergunta 1	As entregas funcionam principalmente pelo iFood (80%), seguido de aplicativo próprio e site (15%), e uma parcela menor via telefone e WhatsApp (5%). Todos os entregadores são fixos, totalizando 7 profissionais, dos quais 6 estão na rua e 1 é responsável por organizar as entregas e fazer as rotas.	As entregas dependem do modo de trabalho do entregador. Alguns entregadores, como eu, fazem todas as entregas de todos os pedidos, independentemente de serem do iFood, telefone ou WhatsApp. Recebemos um valor fixo pelo trabalho e todas as taxas são de nossa responsabilidade, conforme acordado com o restaurante.
Pergunta 2	Os pedidos por iFood representam cerca de 80%, enquanto pedidos via aplicativo/site próprio são aproximadamente 15%, e pedidos via telefone/WhatsApp compreendem cerca de 5% do total.	A quantidade de entregas varia significativamente e é bem distribuída. Em termos percentuais, aproximadamente 33% dos pedidos são feitos por telefone, WhatsApp e iFood, respectivamente.
Pergunta 3	Todos os pedidos são integrados ao sistema. A equipe confere os pedidos no final do dia para garantir que todas as entregas tenham sido feitas corretamente. Qualquer discrepância é abordada com os entregadores.	Os pedidos são impressos, preparados na cozinha e ficam prontos no balcão. Nossa responsabilidade é conferir se há troco e refrigerantes, enquanto a responsabilidade pelo produto em si recai sobre a cozinha ou o responsável pelo empacotamento.

Continua na próxima página

Tabela 4 – Continuação da página anterior

Pergunta	**Felipe**	**Everson**
Pergunta 4	O número de entregas por saída varia de acordo com a demanda e as rotas, podendo variar de 1 a 6 entregas por saída.	O número de entregas por saída é variável, dependendo da rota e da capacidade da caixa. Pode variar de uma única entrega a várias, com situações em que algumas entregas de uma mesma rota ficam a cargo de outro entregador por falta de espaço na caixa.
Pergunta 5	A priorização das entregas é determinada principalmente pelos entregadores, que conhecem bem a região. Não há um aplicativo específico para calcular a rota, mas o estabelecimento oferece orientação sobre a sequência de entregas.	A priorização das entregas não é feita por um aplicativo, mas sim pelos entregadores. Normalmente, pegamos o pedido que está atrasado e ajustamos a rota de acordo com o destino do pedido. Caso não haja outras entregas na mesma direção, o entregador levará uma única entrega.
Pergunta 6	Os entregadores são contratados sob o regime CLT e recebem um salário fixo, além de um adicional de periculosidade. Eles trabalham das 11h às 14h, de segunda a sábado, e fazem de 15 a 20 entregas por dia. Eles são responsáveis por organizar as refeições na mochila antes de sair para as entregas.	O ganho por entrega varia de acordo com o local de trabalho, e a organização da rota é deixada a critério do entregador. Cada entregador organiza sua rota da maneira que considera mais segura para evitar erros ou danos aos produtos entregues.

4.3 Funcionamento do Aplicativo desenvolvido em Python

Finalmente, com todos os algoritmos implementados de maneira satisfatória e integrada, a interface, ou programa, foi finalizado. Na Figura 9 é apresentado a interface do programa final para testes.

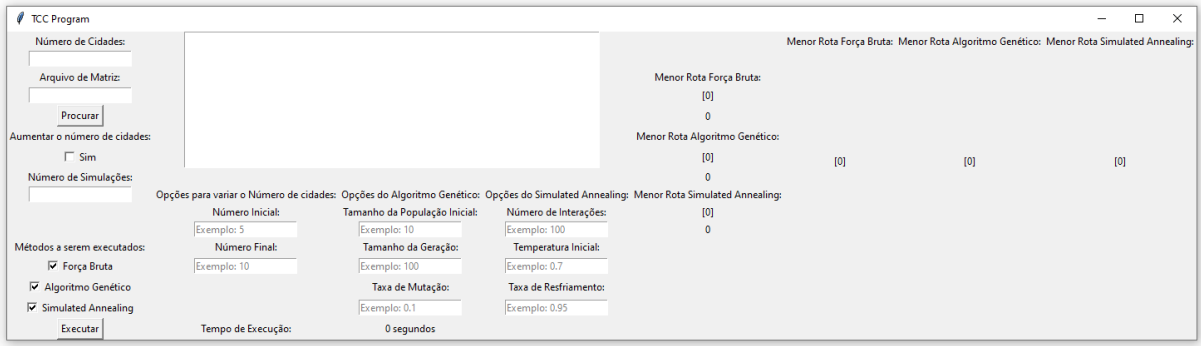


Figura 9 – Imagem da interface final.
Fonte: De autoria própria.

Os conceitos de linha e colunas foram fundamentais para manter uma organização facilmente compreensível na interface, a figura 10 exibe uma execução padrão que o usuário pode inserir.

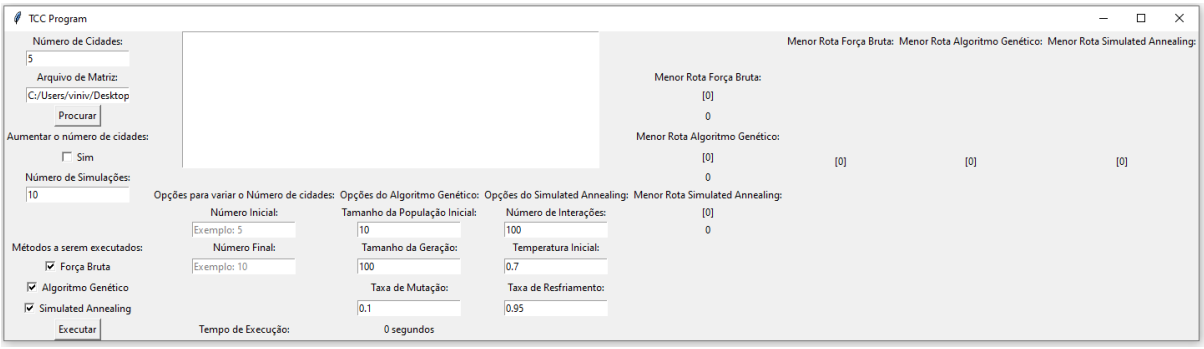


Figura 10 – Imagem de uma entrada padrão feita pelo usuário.
Fonte: De autoria própria.

Após o usuário executar o programa, o mesmo retorna três figuras, sendo a primeira, figura 11, mostra a comparação dos três algoritmos quanto ao tempo de execução conforme as simulações.

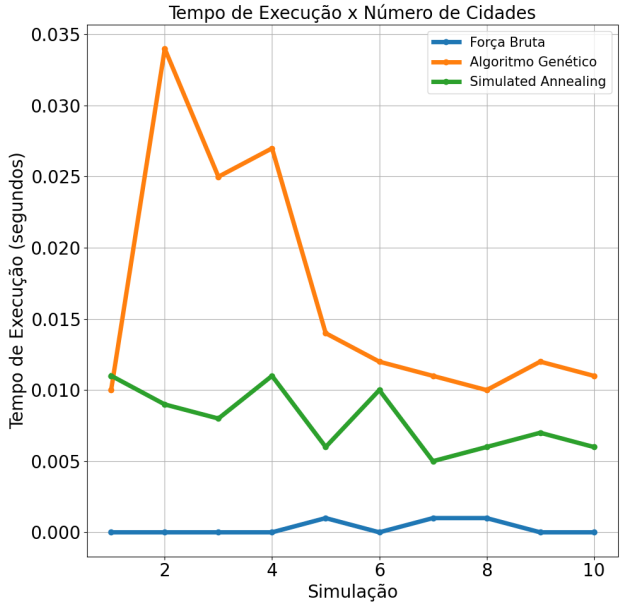


Figura 11 – Gráfico comparativa entre os três métodos.
Fonte: De autoria própria.

A figura 12, mostra o tempo de execução para os dois algoritmos competitivos para o critério de tempo de execução.

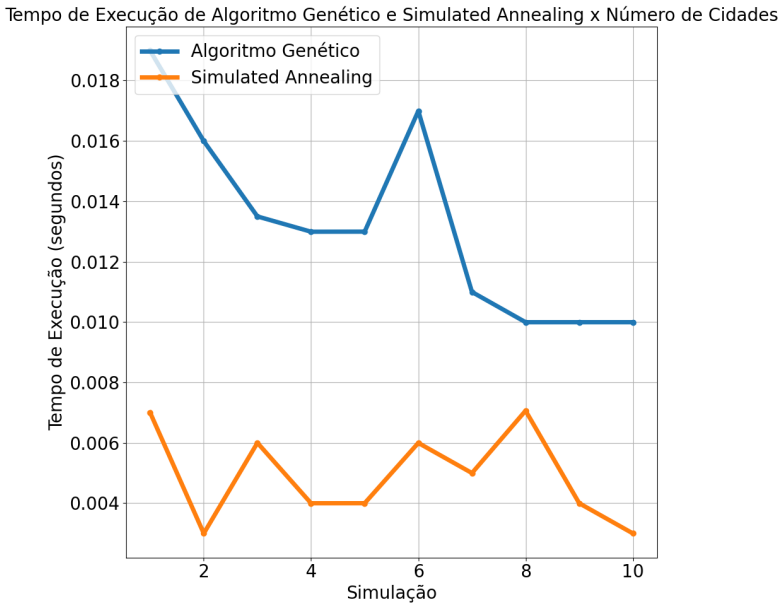


Figura 12 – Gráfico comparativo entre o AG e o SA.
Fonte: De autoria própria.

Já o ultimo gráfico, figura 14, mostra as menores distâncias encontradas para cada método, ao longo das simulações.

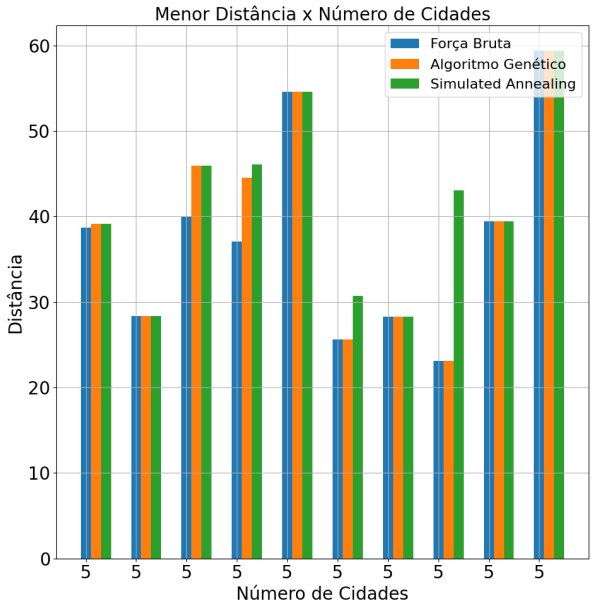


Figura 13 – Gráfico das menores distâncias para cada método ao longo das simulações.
Fonte: De autoria própria.

Por fim, a interface gráfica é atualizada com as principais informações para o usuário, como a menor distância encontrada em cada método e sua respectiva rota, caso a matriz de distâncias contenha quais são os endereços o programa retornará os mesmos organizados, assim como também retorno a média do tempo de execução de cada algoritmo e seu desvio padrão.

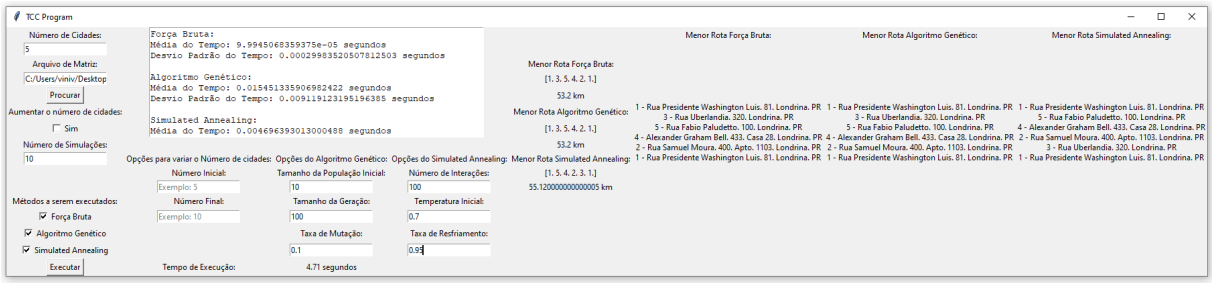
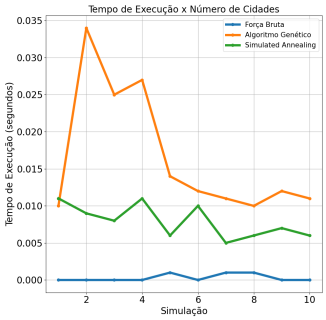
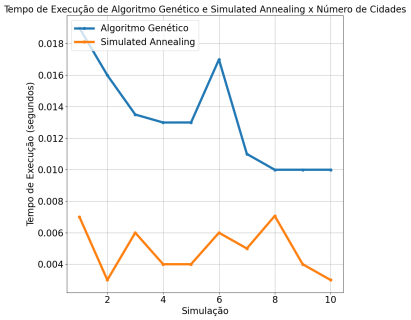


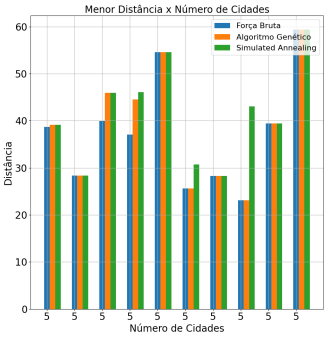
Figura 14 – Resultados entregues na interface gráfica.
Fonte: De autoria própria.



(a) Legenda da figura a



(b) Legenda da figura b



(c) Legenda da figura c

Figura 15 – Legenda geral para as figuras

5 Discussões e Conclusões

5.1 Sugestões de Trabalhos Futuros

Referências

ARAÚJO, W. L. d. *Projeto ideal de suspensão considerando modelo de um quarto de veículo empregando algoritmos genéticos*. Dissertação (B.S. thesis), 2021. 34

BAIDOO, E.; OPPONG, S. O. Solving the tsp using traditional computing approach. *International Journal of Computer Applications*, Foundation of Computer Science, v. 152, n. 8, p. 13–19, 2016. 41

BALLOU, R. *Business Logistics/supply Chain Management: Planning, Organizing, and Controlling the Supply Chain*. Pearson/Prentice Hall, 2004. ISBN 9780130661845. Disponível em: <https://books.google.com.br/books?id=sgsdQAAACAAJ>. 27

BARBOSA, D.; JR., C.; KASHIWABARA, A. Aplicação da otimização por colônia de formigas ao problema de múltiplos caixeiros viajantes no atendimento de ordens de serviço nas empresas de distribuição de energia elétrica. In: *Anais do XI Simpósio Brasileiro de Sistemas de Informação*. Porto Alegre, RS, Brasil: SBC, 2015. p. 23–30. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/sbsi/article/view/5797>. 28

BARBOZA; OLANDOSKI, A. Simulação e técnicas da computação evolucionária aplicadas a problemas de programação linear inteira mista. Centro Federal de Educação Tecnológica do Paraná, 2005. Disponível em: <http://repositorio.utfpr.edu.br:8080/jspui/handle/1/74>. 35

BARRETO, J. M. Inteligência artificial no limiar do século xxi. *Florianópolis: PPP edições*, v. 97, 1999. 35

BENEVIDES, P. F. et al. Aplicação de algoritmos genéticos e simulated annealing para o problema do caixeiro viajante em uma situação real de distribuição de produtos. Brasil, 2011. 15, 28, 29, 36

BISPO, R. C. Planejador de roteiros turísticos: uma aplicação do problema do caixeiro viajante na cidade do recife. Brasil, 2018. Disponível em: <http://repository.ufrpe.br/handle/123456789/722>. 36

CAMPOS, R. S. Desmistificando a inteligência artificial: Uma breve introdução conceitual ao aprendizado de máquina. *Aoristo-International Journal of Phenomenology, Hermeneutics and Metaphysics*, v. 3, n. 1, p. 106–123, 2021. 33

CARVALHO, A. V. d. *O problema do caixeiro viajante com múltiplos passageiros e quota*. Dissertação (Mestrado) — Brasil, 2018. 28

CARVALHO, L. et al. Redução de custo com combustível para uma frota. p. 55–62, 1980. Disponível em: <https://periodicos.set.edu.br/cadernoexatas/article/view/896/725>. 27

CENTER", O. D. *O USO DE IAS E ANÁLISE DE DADOS NA EDUCAÇÃO*. 2023. Acesso em: 20 de fevereiro de 2024. Disponível em: <https://opendatacenter.com.br/blog/o-uso-de-ias-e-analise-de-dados-na-educacao/>. 30

COZMAN, F. G. Inteligência artificial: uma utopia, uma distopia. *TECCOGS: Revista Digital de Tecnologias Cognitivas, São Paulo*, n. 17, p. 32–42, 2018. 32

CRISTINA, A. et al. A TERMODINÂMICA COMO ALGORITMO DE OTIMIZAÇÃO. n. 2015, 2023. 36

Deep Learning Book. *Capítulo 6 – O Perceptron – Parte 1*. 2021. Acesso em: 20 de fevereiro de 2024. Disponível em: <https://www.deeplearningbook.com.br/o-perceptron-parte-1/>. 15, 32

DIAS. *Inteligência Artificial na logística: como funciona e seus benefícios*. 2022. Disponível em: <https://www.trackage.com.br/blog/inteligencia-artificial-na-logistica/>. Acesso em: 01 de março de 2023. 27, 31

EBERHART, R. C. *Neural network PC tools: a practical guide*. [S.l.]: Academic Press, 1990. 31

ecommercebrasil. *Entenda com o Grupo Intelipost como fazer uma roteirização inteligente*. 2023. Disponível em: <https://www.ecommercebrasil.com.br/noticias/grupo-intelipost-roteirizacao-inteligente>. Acesso em: 01 de março de 2023. 31

FERREIRA, I. A. Caixeiro viajante: aplicação da modelagem matemática na otimização de rotas em uma concessionária de energia elétrica. *Revista Produção Online*, v. 20, n. 1, p. 221–246, mar. 2020. Disponível em: <https://www.producaoonline.org.br/rpo/article/view/3491>. 36

FILITTO, D. Algoritmos genéticos: uma visão explanatória. *Revista Multidisciplinar da*, 2008. 35

FONSECA, J. D. d. O. et al. Otimização de rotas de entregas de materiais em uma rede hospitalar por meio do algoritmo do problema do caixeiro viajante. v. 9, 2020. Disponível em: <https://periodicos.uninove.br/revistargss/article/view/16570>. 36

GOUVEIA, L. B. et al. Algoritmos genéticos: aplicando a teoria a um estudo de caso. *Brazilian Journal of Development*, v. 7, n. 3, p. 21053–21077, 2021. 34, 35

HUGO, V. et al. O problema do caixeiro viajante aplicado ao grupo crítico do sistema de entregas de um restaurante. 2017. Disponível em: <http://anais.unespar.edu.br/xi_eepa/data/uploads/artigos/3/3-04.pdf>. 27

KIRKPATRICK, S. et al. Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680, 1983. Disponível em: <https://www.science.org/doi/abs/10.1126/science.220.4598.671>. 36

LACERDA, E. G. de; CARVALHO, A. D. Introdução aos algoritmos genéticos. *Sistemas inteligentes: aplicaçoes a recursos hıdricos e ciências ambientais*, v. 1, p. 99–148, 1999. 34

LIEBERMAN, F. *Introdução à Pesquisa Operacional*. McGraw Hill Brasil, 2010. ISBN 9788563308283. Disponível em: <https://books.google.com.br/books?id=XM_4wAEACAAJ>. 29

LMXlogistica. *MERCADO LOGÍSTICO PÓS-PANDEMIA: O QUE ESPERAR?* 2022. Acesso em: 16 de fevereiro de 2024. Disponível em: <https://www.lmxlogistica.com.br/mercado-logistico-pos-pandemia-o-que-esperar/#:~:text=Principais%20mudanÃğas%20na%20logÃŋstica%20pÃşs-pandemia&text=A%20crise%20tem%20contribuÃŋdo%20para,transporte%20ferroviÃąrio%2C%20rodoviÃąrio%20e%20aÃľreo.> 27

MARTINS, A. L. Potenciais aplicações da inteligência artificial na ciência da informação. *Informação & Informação*, v. 15, n. 1, p. 1–16, 2010. 33

MCCARTHY, J. WHAT IS ARTIFICIAL INTELLIGENCE ? p. 1–15, 2007. 30

Mobilidade Sampa. *Carnaval 2023: como o feriado impacta nos processos de logística?* 2023. Disponível em: <https://mobilidadesampa.com.br/2023/02/carnaval-2023-como-o-feriado-impacta-nos-processos-de-logistica/>. Acesso em: 01 de março de 2023. 27

MOREIRA, I. I. P. et al. Redes neurais artificiais para previsão de capacidade de carga em estacas do tipo hélice contínua artificial neural networks for load capacity prediction in continuous flight auger piles. *Brazilian Journal of Development*, v. 7, n. 12, p. 112577–112597, 2021. 33

NASCIMENTO, D. B. et al. Análise comparativa de algoritmos heurísticos para resolução do problema do caixeiro-viajante em grafos não clusterizados. 2004. Disponível em: <https://abepro.org.br/biblioteca/ENEGEP2004_Enegep0601_0567.pdf>. 36

PACHECO, M. A. C. et al. Algoritmos genéticos: princípios e aplicações. *ICA: Laboratório de Inteligência Computacional Aplicada. Departamento de Engenharia Elétrica. Pontifícia Universidade Católica do Rio de Janeiro. Fonte desconhecida*, v. 28, 1999. 19, 34

PINHEIRO, M.; OLIVEIRA, H. Inteligência artificial: Estudos e usos na ciência da informação no brasil. *Revista Ibero-Americana de Ciência da Informação*, v. 15, n. 3, p. 950–968, 2022. 30

PINHEIRO, M. D. M. et al. Perceptron multicamadas: uma ferramenta de aprendizado supervisionado. Niterói, 2021. 32

PRESTES Álvaro. Uma análise experimental de abordagens heurísticas aplicadas ao problema do caixeiro viajante. 2006. Disponível em: <https://repositorio.ufrn.br/handle/123456789/17962>. 29

QUARESMA, A. Inteligência artificial fraca e força bruta computacional. *TECHNO REVIEW. International Technology, Science and Society Review/Revista Internacional De Tecnología, Ciencia Y Sociedad*, v. 10, n. 1, p. 67–78, 2021. 32

Roberto N. Onody. *Teste de Turing e Inteligência Artificial*. 2021. Acesso em: 16 de fevereiro de 2024. Disponível em: <https://www2.ifsc.usp.br/portal-ifsc/teste-de-turing-e-inteligencia-artificial/.> 31

ROBINSON. *On the Hamiltonian Game*. 1949. Disponível em: <https://web.archive.org/web/20200629071813/https://apps.dtic.mil/dtic/tr/fulltext/u2/204961.pdf>. Acesso em: 01 de março de 2023. 28

RUSSELL, S.; NORVIG, P. *Inteligência Artificial*. Rio de Janeiro: Elsevier, 2004. 34

SALMEN, C. S.; WACHOWICZ, M. A atribuição da pessoa jurídica à inteligência artificial: desafios e sua efetividade the attribution of the legal entity to artificial intelligence: challenges and its effectiveness. *Brazilian Journal of Development*, v. 7, n. 7, p. 71438–71457, 2021. 32, 33

SEARLE, J. R. *Mente, Cérebro e Ciência*. Lisboa: Edições 70, 1987. 33

SILVA, G. A. N. d. et al. Algoritmos heuristicos contrutivos aplicados ao problema do caixeiro viajante para definição de rotas otimizadas. *Colloquium Exactarum. ISSN: 2178-8332*, v. 5, n. 2, p. 30–46, nov. 2013. Disponível em: <https://journal.unoeste.br/index.php/ce/article/view/939>. 29

SILVA, G. N.; ARRUDA, N. Teste de turing: Um computador é capaz de pensar? In: *CONGRESSO NACIONAL DE PESQUISA E ENSINO EM CIÊNCIAS (CONAPESC)*. [S.l.: s.n.], 2016. 31

SILVA, R. O. da; SILVA, I. R. S. Linguagem de programação python. *TECNOLOGIAS EM PROJEÇÃO*, v. 10, n. 1, p. 55–71, 2019. 39

SILVEIRA da et al. Ensaio teórico sobre o uso das redes neurais artificiais no gerenciamento de resultados. *Revista de Gestão e Secretariado (Management and Administrative Professional Review)*, v. 14, n. 1, p. 913–931, 2023. 33

SOUZA, S. S. F.; ROMERO, R. Algoritmo Imunológico Artificial CLONALG e Algoritmo Genético Aplicados ao Problema do Caixeiro Viajante Introdução. v. 2, p. 1–6, 2014. Disponível em: <https://proceedings.sbmac.org.br/sbmac/article/view/307/309>. 28

VÁZQUEZ-GUZMÁN, G.; MARTÍNEZ-RODRÍGUEZ, P. R.; SOSA-ZÚÑIGA, J. M. Inversor monofásico de alta eficiencia sin transformador para aplicaciones fotovoltaicas. *Ingeniería, investigación y tecnología*, Facultad de Ingeniería, UNAM, v. 16, n. 2, p. 173–184, 2015. 34

WANG, K. Logistics 4.0 solution-new challenges and opportunities. In: ATLANTIS PRESS. *6th international workshop of advanced manufacturing and automation*. [S.l.], 2016. p. 68–74. 27

ZADEH, L. A. Information and control. fuzzy sets. *North-Holland*, v. 8, n. 3, p. 15–25, 1965. 33

APÊNDICE A - Código Implementado do Método Busca Exaustiva em Matlab

```
1
2
3  function FB_func = forcabruta(MatrizDistTrab)
4
5  tempoFB = tic;
6  DistMatriz = MatrizDistTrab;
7
8  VetorIndi = 1:size(DistMatriz, 1);
9
10 % Permutacao dos indices das cidades
11 Pi = perms(VetorIndi(2:end));
12
13 % Adicionar a cidade de partida no inicio de cada permutacao
14 Pi = [ones(size(Pi, 1), 1), Pi, ones(size(Pi, 1), 1)];
15
16 % Variavel da Distancia
17 distanciamenor  = inf;
18 distanciamaior = 0;
19
20 % Calculo da rota
21 VetorDistancias = zeros(length(Pi),1);
22 Indice_Rota_menor = 0;
23 Indice_Rota_maior = 0;
24
25 for i = 1:length(Pi)
26     rota = Pi(i, :); % Armazena a rota atual
27     distancia   = 0;
28
29     for j = 1:length(rota)
30         Cidade_partida = rota(j);
31         Cidade_chegada = rota(mod(j, length(rota)) + 1);
32         distancia = distancia + DistMatriz(Cidade_partida, Cidade_chegada);
33     end
34
35     % Armazenador de distancias
36     VetorDistancias(i,1) = distancia;
37
```

```
38      % Comparadores
39      if distancia < distanciamenor
40          distanciamenor = distancia;
41          Indice_Rota_menor = i;
42      end
43      if distancia > distanciamaior
44          distanciamaior = distancia;
45          Indice_Rota_maior = i;
46      end
47
48  end
49
50  % Obtem a rota com menor distancia (desconsiderando a cidade de partida
        duplicada)
51  Rota_menor = Pi(Indice_Rota_menor, :);
52
53  % Obtem a rota com maior distancia (desconsiderando a cidade de partida
        duplicada)
54  Rota_maior = Pi(Indice_Rota_maior, :);
55  tempoFB = toc(tempoFB);
56  % Preparando a saida da funcao
57  FB_func.distanciamenor = distanciamenor;
58  FB_func.Indice_Rota_menor = Indice_Rota_menor;
59  FB_func.Rota_menor = Rota_menor;
60  FB_func.distanciamaior = distanciamaior;
61  FB_func.Indice_Rota_maior = Indice_Rota_maior;
62  FB_func.Rota_maior = Rota_maior;
63  FB_func.tempoFB = tempoFB;
64  %% Exibi as distancias
65  % disp(['A distancia total da menor rota eh e a rota eh a:']);
66  % distanciamenor
67  % Indice_Rota_menor
68  % Rota_menor
69  % disp(['A distancia total da maior rota eh a rota eh a:']);
70  % distanciamaior
71  % Indice_Rota_maior
72  % Rota_maior
73
74
75
76  end
```

Listing 1 – Código em Matlab Da Busca Exaustiva

APÊNDICE B - Código Implementado do Método Algoritmo Genético em Matlab

```
1
2
3
4  function AG_Func = alggenetico(MatrizDistTrab,tam_Pop_ini,tam_gera)
5
6  tempoAG = tic;
7  DistMatriz = MatrizDistTrab;
8
9  numCidades = length(DistMatriz);
10
11 % Alocacao de variaveis para aumentar a velocidade
12 Populacao = zeros(tam_Pop_ini, (numCidades+1));
13 FilhosPai = zeros(4, (numCidades+1));
14 FilhosMae = zeros(4, (numCidades+1));
15 nmr_filhos = zeros((tam_Pop_ini-2),(numCidades+1));
16
17 % Gere a populacao inicial contendo todas as cidades, mas em ordem
       aleatoria
18 Percurso = zeros(tam_Pop_ini, numCidades+1); % +1 para incluir a cidade de
       origem no fim da rota
19
20 %Geracao da Populacao Inicial
21 for k = 1:tam_Pop_ini
22
23     % Gera um vetor de cidades (exceto a cidade de origem) em ordem
       aleatoria
24     cidades_aleatorias = randperm(numCidades-1) + 1;
25
26     % Cria o cromossomo (rota) com a cidade de origem no inicio e fim
27     Populacao(k, :) = [1, cidades_aleatorias, 1];
28
29
30 end
31
32
33 for g = 1:tam_gera
34
```

```
35
36      %Calcula o percurso total de cada vetor solucao
37      for k = 1:size(Populacao,1)
38          Percurso(k, g) = Calc_Dist(DistMatriz, Populacao(k,:));
39      end
40
41      %Escolhe os mais aptos
42      clear a b;
43      [a, b] = sort(Percurso(:,g));
44
45      % Recupera a menor distancia encontrada
46      menorDistancia = a(1);
47
48      % Recupera a melhor rota encontrada
49      melhorRota = Populacao(b(1,1), :);
50
51      Pai = Populacao(b(1,1), :);
52      Mae = Populacao(b(2,1), :);
53
54      %   Calcula o Nmr de filhos
55      num_filhos = size(nmr_filhos,1);
56
57      if rem(num_filhos, 2) == 0
58          %   Numero de filhos eh par, logo ambos possuem a mesma quantidade
59          palim = num_filhos/2;
60          malim = num_filhos/2;
61
62      else
63          %   Numero de filhos eh impar, logo sortea um genitor para ter mais
64          %   filhos
65          escolha = randi([1,2]);
66
67          if escolha == 1
68              %   Pai com mais filhos
69              palim = (num_filhos + 1) / 2;
70              malim = (num_filhos - 1) / 2;
71
72          else
73              %   Mae com mais filhos
74              palim = (num_filhos - 1) / 2;
75              malim = (num_filhos + 1) / 2;
76          end
77      end
78
79      %Cada Genitor vai gerar 2 filhos por permuta
80      %Sorteia e permuta as duas posicoes do Pai
81      for pa = 1:palim
```

```
82          clear a b;
83          [~, b] = sort(rand(2,numCidades));
84          %b = b + 1;
85
86          FilhosPai(pa, :) = Pai;
87          FilhosPai(pa, b(1, 1)) = Pai(1, b(1, 2));
88          FilhosPai(pa, b(1, 2)) = Pai(1, b(1, 1));
89      end
90
91      %Sorteia e permuta as duas posicoes da Mae
92      for ma = 1:malim
93          clear a b;
94          [~, b] = sort(rand(2,numCidades));
95          %b = b + 1;
96
97          FilhosMae(ma, :) = Mae;
98          FilhosMae(ma, b(1, 1)) = Mae(1, b(1, 2));
99          FilhosMae(ma, b(1, 2)) = Mae(1, b(1, 1));
100     end
101
102
103     %Atualiza a Populacao
104     Populacao = [Pai; Mae; FilhosPai; FilhosMae];
105
106 end
107
108 tempoAG = toc(tempoAG);
109 %plot(min(Percurso, [], 1));
110
111
112 % Preparando a saida da funcao
113 AG_Func.tempoAG = tempoAG;
114 AG_Func.menorDistancia = menorDistancia;
115 AG_Func.melhorRota = melhorRota;
116
117
118
119
120
121
122
123 end
```

Listing 2 – Código em Matlab Do Algoritmo Genético

APÊNDICE C - Código Implementado do Método Simulated Annealing em Matlab

```
1
2  function SA_Func = simulatedAnnealing(matrizDistancia, temperaturaInicial,
       taxaResfriamento, numIteracoes)
3
4      tic_total = tic;
5
6      %   Definindo variaveis
7      numCidades          = size(matrizDistancia, 1);
8
9      %   Gera a solucao inicial aleatoria
10     cidades_aleatorias = randperm(numCidades-1) + 1;
11     melhorRota          = [1, cidades_aleatorias, 1]; %  Solucao inicial
       Aleatoria
12     melhorCusto         = calcularCustoRota(melhorRota, matrizDistancia);
13
14     %   Variaveis para parametro de comparacao
15     rotaAtual = melhorRota;
16     custoAtual = melhorCusto;
17
18     for iteracao = 1:numIteracoes
19
20         % Gera uma nova solucao vizinha alterando aleatoriamente a solucao
       atual
21         novaRota = gerarNovaRotaVizinha(rotaAtual);
22         novoCusto = calcularCustoRota(novaRota, matrizDistancia);
23
24         % Calcula a diferenca de custo entre a nova solucao e a solucao
       atual
25         deltaCusto = novoCusto - custoAtual;
26
27         % Aceita a nova solucao se ela for melhor ou com uma certa
       probabilidade se for pior
28         if (deltaCusto < 0) || (rand() < exp(-deltaCusto /
       temperaturaInicial))
29             rotaAtual = novaRota;
30             custoAtual = novoCusto;
31
```

```
32              % Atualiza a melhor solucao encontrada
33              if custoAtual < melhorCusto
34                  melhorRota = rotaAtual;
35                  melhorCusto = custoAtual;
36              end
37          end
38
39          % Atualiza a temperatura
40          temperaturaInicial = temperaturaInicial * taxaResfriamento;
41      end
42
43      tempoSA = toc(tic_total);
44
45      % Preparando a saida da funcao
46      SA_Func.tempoSA = tempoSA;
47      SA_Func.menorDistancia = melhorCusto;
48      SA_Func.melhorRota = melhorRota;
49
50
51
52  end
53
54  function custo = calcularCustoRota(rota, matrizDistancia)
55
56      % Calcula o custo total da rota
57      custo = 0;
58      numCidades = length(rota);
59
60      for i = 1:numCidades
61
62          cidadeAtual = rota(i);
63          cidadeProxima = rota(mod(i, numCidades) + 1); % Proxima cidade
        considerando rota circular
64          custo = custo + matrizDistancia(cidadeAtual, cidadeProxima);
65
66      end
67  end
68
69  function novaRota = gerarNovaRotaVizinha(rotaAtual)
70
71      % Gera uma nova solucao vizinha alterando aleatoriamente a solucao
        atual
72      numCidades = length(rotaAtual);
73      indice1 = randi(numCidades);
74      indice2 = randi(numCidades);
75      novaRota = rotaAtual;
```

```
76      novaRota([indice1, indice2]) = novaRota([indice2, indice1]); % Troca
        duas cidades aleatoriamente
77
78  end
```

Listing 3 – Código em Matlab Do Simulated Annealing

APÊNDICE D - Código Implementado do Método Busca Exaustiva em Python

APÊNDICE F - Código Implementado do Método Algoritmo Genético em Python

APÊNDICE G - Código Implementado do Método Simulated Annealing em Python