USO DE ALGORITMO GENÉTICO PARA ANÁLISE DE TRAJETOS E PERCURSOS

Mateus Platinetty POMPERMAYER
1901305@escolas.anchieta.br
Bach. em Ciência da Computação, Centro Universitário Anchieta

Samuel Mayer RUFINO
2004245@escolas.anchieta.br
Bach. em Ciência da Computação, Centro Universitário Anchieta

Vinicius Pinto GUEDES
1901162@escolas.anchieta.br
Bach. em Ciência da Computação, Centro Universitário Anchieta

Clayton Augusto VALDO
clayton.valdo@anchieta.br
Orientador, Centro Universitário Anchieta, UniAnchieta

Resumo

O presente artigo tem como objetivo a criação de um sistema para atender o setor de logística, partindo da teoria da seleção natural, o projeto é composto por algoritmo genético, problema do caixeiro viajante, inteligência artificial e cruzamento *order crossover*, codificado na linguagem *python*. O artigo descreve todo embasamento teórico dos conceitos utilizados para desenvolvimento do sistema e todo detalhamento do código para melhor entendimento dos cálculos realizados para a solução do problema central do estudo. A partir da finalização do projeto, entendeu-se a importância do projeto para o ramo, com impacto no cliente final e processos dentro da empresa.
Palavras-Chave:

inteligência artificial; algoritmo genético; python.

Abstract

This article aims to create a system to serve the logistics sector, based on the theory of natural selection, the project is composed of a genetic algorithm, the traveling salesman problem, artificial intelligence and order cross-over, coded in the python language. The article describes all the theoretical basis of the concepts used to develop the system and all the details of the code for a better understanding of the calculations performed to solve the central problem of the study. From the finalization of the project, the im-portance of the project for the branch was understood, with impact on the end customer and processes within the company.

Keywords:

artificial intelligence; genetic algorithm; python.

INTRODUÇÃO

Quando uma empresa tenta reduzir o seu estoque - e para isso efetua pedidos menores, porém com uma maior frequência - acaba causando um número de entregas muito superior e um problema na localização das entregas. Por isso as transportadoras precisam garantir uma frota capacitada para um número de entregas grande e, com isso, um maior aproveitamento nos percursos que fará até a entrega das mercadorias, sempre pensando em prazos com relação à data e horários. Se não existe um bom transporte, o processo de entregas fica comprometido (LOPÉZ, 2005, p.41).

Este trabalho está focado no uso de algoritmo genético para atender uma transportadora, onde foi realizado um estudo para entender os problemas enfrentados no cotidiano de uma empresa de logística, com destaque para a resolução de problemas ou melhorias do sistema já utilizado. Após análise do cenário, entendeu-se que havia uma melhora caso se utilizasse inteligência artificial, pois notou-se que algumas tomadas de decisão para rota de transporte de seus veículos eram com base em dados superficiais, como distâncias de um ponto de partida e de um ponto de chegada, não levando-se em consideração todas as possibilidades de percurso, pela tarefa ser executada por um colaborador e não por uma máquina.

Assim, é possível agregar ao sistema uma análise das possibilidades de percursos, com os todos os dados a serem analisados, definindo melhor rota, tornando um caminho eficiente para a entrega final. Com isso, entende-se que o uso dessa tecnologia no sistema da empresa pode ser benéfico, para definição de novas rotas, menor consumo de tempo para realizar a análise da rota, mais conforto aos colaboradores que realizam os transportes, menor consumo de combustível e menos impacto aos veículos por sempre realizarem a menor rota.

DESENVOLVIMENTO

A tecnologia sempre está acompanhando as evoluções naturais, principalmente a dos seres humanos, grande parte das invenções e realizações na área é baseada no que é apreendido no mundo real e levado para a tecnologia.

Para o desenvolvimento do trabalho foi realizado um estudo em uma empresa multinacional de logística. Com isso, foi notado um método de execução de tomadas de decisão de rotas de entregas para seus clientes finais, como um problema de processos. Assim, o grupo atuou na criação de uma *feature*, utilizando algoritmo genético junto com a utilização do problema do caixeiro viajante, para a criação de uma inteligência artificial.

Inteligência Artificial

Taulli (2020, p.17) argumenta que para entender a inteligência artificial (IA) temos que citar o Allan Turing, que pode ser considerado o "pai da IA". Allan Turing estabeleceu basicamente os conceitos de uma inteligência artificial e como classificar se uma máquina é ou não inteligente. Para entender a classificar, ele criou o "teste de Turing", um jogo com três participantes, dois humanos e uma máquina. O avaliador, humano, tem como objetivo descobrir qual é a máquina e qual é o humano, se for incapaz de identificar, pode-se dizer que é uma "máquina inteligente".

Machine Learning é uma importante área para entender o funcionamento de uma inteligência artificial, como define Silva (2021): "Quando estamos criando um modelo de aprendizado de máquina, não informamos ao computador os passos a seguir para que ele aprenda o que precisa, isso porque o conhecimento é adquirido, [...]".

Dessa forma é uma máquina com capacidade de aprendizado sem um programador ensinando o que deve ser realizado. Todos esses modelos criados, e que vêm sendo utilizados, são com base em redes neurais humanas, podemos dizer que usamos redes neurais artificias para que um aprendizado possa ocorrer. Como escreve Carvalho (2000): "Redes Neurais Artificiais são técnicas computacionais que apresentam um modelo matemático inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência".

Pode-se compreender que inteligência artificial é um conceito criado a partir do aprendizado artificial, baseado em como os seres humanos aprendem usando a mesma estrutura de uma rede neural real.

Algoritmo Genético

Algoritmos genéticos, como cita Pacheco (1999, p.1), "São algoritmos probabilísticos que fornecem um mecanismo de busca paralela e adaptativa baseado no princípio de sobrevivência dos mais aptos e na reprodução". Com isso, podemos definir que esses algoritmos são baseados na teoria da evolução por seleção natural, proposta por Charles Darwin e Alfred Wallace, que vem ganhando mais evidências e aprimoramentos desde a sua publicação (CASTRO, 2022, p.186, apud AYALA, 2008).

Santos (2015, p. 47) cita seleção natural da seguinte forma: "Seleção natural é reprodução diferencial por conta de variações na capacidade de sobrevivência das populações de uma espécie em um determinado ambiente. Esse processo pode levar ao aumento na proporção das características hereditárias vantajosas entre uma geração e a próxima."

Pode-se entender que o algoritmo tem embasamento na teoria da evolução mais aceita pela ciência e utiliza conceitos dela para seu desenvolvimento e confirmação de resultados. O algoritmo desenvolvido está descrito neste artigo, na sessão Projeto.

Problema do caixeiro viajante

O problema do caixeiro viajante, para Silveira (2000), é clássico na computação, sendo um exemplo de problema de otimização combinatória.

Araripe (2017, p.10, APUD LISBOA, 2007) define o problema: "Também conhecido por *Travelling Salesman Problem* (TSP), o PCV tem por objetivo basicamente fazer com que todos os clientes sejam visitados apenas uma vez e que a distância total percorrida entre eles seja minimizada, ou seja, o roteiro gerado seja o menor possível."

Com o cenário em mente, é possível partir para a solução do problema de diversas formas, e com algoritmos diferentes, Silveira (2000) cita que uma das primeiras formas para se resolver é enumerar todos os caminhos possíveis e analisar qual menor caminho a se tomar.

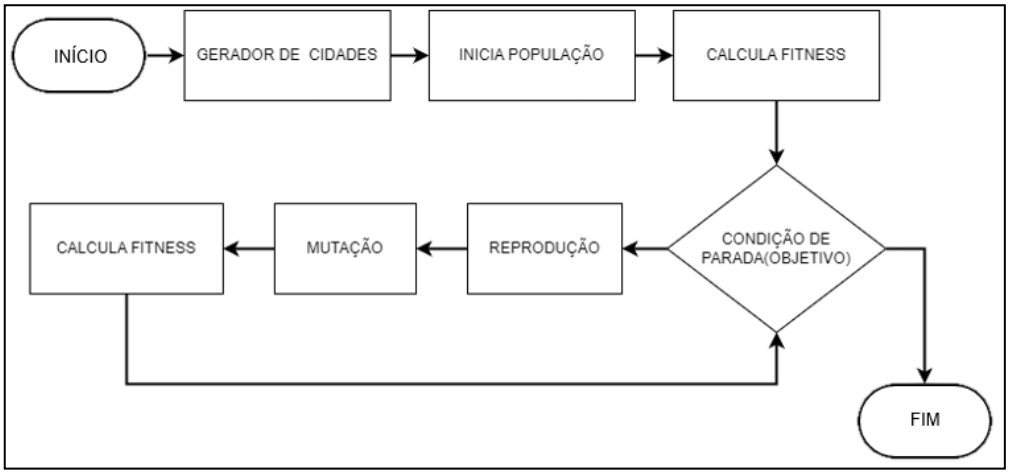
O grupo desenvolveu um algoritmo junto com algoritmo genético para resolver o problema do caixeiro viajante, e com inteligência artificial ser possível realizar uma tomada de decisão.

PROJETO

A partir dos conceitos da teoria da evolução por seleção natural e estudo do algoritmo genético, foi desenvolvido um projeto na linguagem de programação *Python*, versão 3.10.5, utilizando as bibliotecas (*Random, Math* e *CustomKinder*). Foi utilizado o conceito de gene e cromossomo adaptados ao projeto, sendo, gene o index de uma cidade e cromossomo, uma lista que contém a sequência de cidades, ou seja, é o caminho que aquele indivíduo vai seguir. Para o desenvolvimento, foi criado um fluxograma para entender as partes do programa a serem realizadas.

O fluxograma desenvolvido e ilustrado abaixo é dividido em sete etapas.

Figura 1. Fluxograma do projeto.



Gerador de cidades, esta etapa gera os dados necessários para a execução da IA. O algoritmo consome uma lista (tipo de dado da linguagem Python) contendo as ligações entre as "cidades" e suas respectivas distâncias. A tabela 1 apresenta um exemplo de uma geração de quatro cidades:

Tabela 1. Exemplo do gerador de cidades

CIDADE	1	2	3	4
1	0	51	72	68
2	51	0	48	18
3	72	48	0	33
4	68	18	33	0

Na Tabela 1, pode-se visualizar que a distância entre a cidade três e a cidade dois é de quarenta e oito. Coordenadas preenchidas com o valor zero significa que não é possível viajar por este caminho.

Inicia população, configura-se o tamanho da população e ao chegar nesta etapa os indivíduos dessa população são gerados, atribuindo um valor inicial de cromossomo para cada indivíduo.

Calcula fitness, executa o cálculo de *fitness* para cada indivíduo da população. Para cada indivíduo é executado:

fitness = fitness + distâncias[cromossomo][cromossomo do próximo individuo]

O valor *fitness* se inicia em zero e após incrementar a distância de todas as cidades, utilizando o cálculo acima, temos o valor *fitness*.

Condição de parada, esse é o objetivo da inteligência artificial, no algoritmo desenvolvido o objetivo é definido por um limite de gerações, após as gerações geradas serem maiores que o limite definido, o algoritmo termina a execução.

Reprodução, para realizar a reprodução foi utilizado o algoritmo cruzamento *Order Crossover*, que simula a reprodução sexuada que acontece no processo natural, na qual dois pais combinados geram filhos com parte das suas características.

Mutação, para que aconteça a mutação, a geração é passada por um sorteio para cada indivíduo da população, caso o indivíduo seja sorteado é escolhido dois genes aleatórios do cromossomo, invertendo esses genes entre si, ou seja, o gene um troca de posição com o gene dois. O sorteio

realizado tem a chance de execução dele em 5%, essa taxa foi escolhida para que o cromossomo de bons indivíduos não seja prejudicado. Brito (2006) define a taxa de mutação em 1%, porém para o algoritmo desenvolvido essa taxa não foi suficiente, pois foi observado pouca mudança dos indivíduos conforme as gerações passavam.

Codificação

O código está disponível para ser baixado na íntegra pelo link:

https://github.com/smrsassa/ag-caixeiro-viajante

No código 1, "instancia-se" a classe evolução, a *main loop* da inteligência artificial do projeto. Na linha quatorze do código, define-se uma condição para quando executado diretamente pelo terminal, a saída do sistema seja mais detalhada.

caixeiroViajante.py

```python
from src.evolucao import Evolucao


class CaixeiroViajante:
    def __init__(self, qtdeCidade, tamanhoPopulacao, limiteGeracoes, cidadeInicial) -> None:
        self.evolucao = Evolucao(qtdeCidade, tamanhoPopulacao, limiteGeracoes, cidadeInicial)

    def getGeracaoCidades(self) -> list:
        return self.evolucao.getGeracaoCidades()

    def run(self, log = True) -> list:
        return self.evolucao.evoluir(log)

if __name__ == '__main__':
    caixeiroViajante = CaixeiroViajante(10, 6, 100, 3)
    caixeiroViajante.run()
```

Código 1. caixeiroViajante.py

No código abaixo, observa-se o uso da biblioteca *customkinter*, essa biblioteca é consumida pelos métodos da classe App para criação da interface gráfica do sistema, os métodos têm as seguintes funcionalidades:

Para iniciar a interface gráfica do sistema é consumido o método *__init__*, com o uso do sistema é acionado o *button_event*, com a característica de validar as informações escritas pelo utilizador por meio da função *validarInput*, pois o sistema só aceita valores numéricos, após essa verificação aciona-se a classe *CaixeiroViajante*, com as passagens de parâmetros necessárias. Ao retornar, cria-se as cidades no *grid* da interface com o *desenhaCidade*, que por sua vez calcula os pontos a serem colocados e as insere em tela, com uma mudança estética do *create_circle*, método responsável

unicamente para isso. Os métodos *change_appearance_mode* e *on_closing*, têm como objetivo novas funcionalidades visuais para o utilizador.

Código 2. app.py

```
app.py > ...
1   import tkinter
2   import tkinter.messagebox as mb
3   import customtkinter
4   from app import CaixeiroViajante
5
6
7   customtkinter.set_appearance_mode("System")
8   customtkinter.set_default_color_theme("blue")
9
10  class App(customtkinter.CTk):
11      WIDTH = 1024
12      HEIGHT = 640
13
14  >   def __init__(self) -> None:···
105
106 >   def create_circle(self, x, y, r, canvas) -> None:···
112
113 >   def desenhaCidade(self, cidades) -> None:···
134
135 >   def validarInput(self) -> bool:···
147
148 >   def button_event(self) -> None:···
179
180 >   def change_appearance_mode(self, new_appearance_mode) -> None:···
182
183 >   def on_closing(self, event=0) -> None:···
185
186 if __name__ == "__main__":
187     app = App()
188     app.mainloop()
```

O código abaixo é a parte de criação de um modelo de cidade, para localizá-la em um espaço no plano cartesiano, o método *setCoordenada* gera uma posição aleatória para a cidade.

Código 3. cidade.py

```
src > cidade.py > ...
1   import random
2
3   class Cidade:
4       def __init__(self) -> None:
5           self.pontoX = 0
6           self.pontoY = 0
7           self.id = 0
8
9       def setCoordenada(self) -> None:
10          self.pontoX = random.randint(1,100)
11          self.pontoY = random.randint(1,100)
```

Código 4, mostra a classe que executa o fluxograma apresentado na sessão Projeto deste documento, quando iniciado o construtor *__init__*, crias as "instâncias" necessárias para a execução dos cálculos do sistema. O método "evoluir" executa o laço principal do sistema, acionando o necessário para reprodução, mutação e cálculo de *fitness*. Para auxiliá-lo existe *getGeracaoCidades*, que é utilizado pela interface para exibir ao utilizador, *definirParada*, verifica se a geração atual ultrapassou a quantidade de gerações limite e *mutacao*, inverte dois genes de indivíduos selecionados aleatoriamente.

Código 4. evolucao.py

```
src > evolucao.py > ...
1   import random
2   from src.geradorCidades import GeradorCidades
3   from src.populacao import Populacao
4   from src.reproducao import Reproducao
5
6
7   class Evolucao:
8       def __init__(self, qtdeCidade, tamanhoPopulacao, limiteGeracoes, cidadeInicial) -> None: ···
16
17      def getGeracaoCidades(self) -> list: ···
19
20      def definirParada(self) -> bool: ···
22
23      def mutacao(self) -> None: ···
32
33      def evoluir(self, log = True) -> list: ···
60
```

Para o código 5, ele é capaz de criar as cidades consumindo a classe Cidade, criando os atributos necessários de uma cidade e calculando a distância entre elas, utilizando Pitágoras.

Código 5. geradorCidades.py

```
src > geradorCidades.py > ...
1   from src.cidade import Cidade
2   import math
3
4
5   class GeradorCidades:
6       def __init__(self, qtdeCidade) -> None: ···
12
13      def criarCidades(self) -> None: ···
19
20      def definirDistancias(self) -> None: ···
28
29
30  if __name__ == "__main__":
31      teste = GeradorCidades(10)
32      for linha in teste.distancias:
33          print(linha)
```

No código de individuo, após o construtor *__init__*, inicia-se os atributos da classe, os métodos executam mudanças necessárias no individuo, sendo *getCromossomo* a cópia da lista de cromossomo dele, *cromossomoInicial* gera o primeiro cromossomo de um indivíduo, sendo uma sequência de cidades aleatórias e o *inverterGene*, recebe dois genes como parâmetro e inverte a posição entre eles.

Código 6. individuo.py

```
src > individuo.py > ...
1   import random
2
3
4   class Individuo:
5       def __init__(self) -> None: ···
8
9       def getCromossomo(self) -> list: ···
11
12      def cromossomoInicial(self, qtdeCidades, cidadeInicial) -> None: ···
19
20      def inverterGene(self, gene1, gene2) -> None:
21          self.cromossomo[gene1], self.cromossomo[gene2] = self.cromossomo[gene2], self.cromossomo[gene1]
```

A população tem como objetivo gerenciar todos os indivíduos e após a conclusão exibir o melhor entre eles, baseado no maior *fitness*, ao iniciar uma população, o construtor *__init__* inicia os atributos da classe, para verificar se o tamanho da população é válido utiliza-se o *valorTamanho*, após, utiliza-se o método *iniciaPopulacao*, "instanciando" todos os indivíduos e os inclui na lista de indivíduos da população.

A população passa por vários processos durante o laço da evolução, *calcularFitness* calcula o quão bom são todos os indivíduos da população, somando as suas distâncias, e a lista calculada é ordenada de melhor para pior, *addFilhos* adiciona dois indivíduos à população e o método *ajustarPopulacao* apaga da lista de indivíduos todos que excedem o tamanho da população, excluindo os menos

eficientes, este processo é a parte de algoritmo genético, pois quem pior se adapta é eliminado. Conforme cada finalização de ciclo, a geração incrementa em um, por meio da função *valorGeracao*. Os métodos *getIndividuos* e *ajustarPopulacao* servem para exibição de resultados, sendo um para terminal e um para interface gráfica. Abaixo, o código que representa o algoritmo de população:

Código 7. populacao.py

Para o código abaixo, define-se a classe de reprodução com três métodos, sendo *__init__* método construtor para definição de variáveis, *melhoresIndividuos* retorna os dois primeiros indivíduos da população, consequentemente retorna os dois melhores, pois eles estão ordenados por seu *fitness* e o método "reproduzir", que executa o cruzamento *order crossover(OX)*, gerando dois novos indivíduos a partir dos melhores indivíduos da população.

Código 8. reproducao.py

```
src > reproducao.py > ...
1   import random
2
3
4   class Reproducao:
5       def __init__(self) -> None: ···
7
8       def melhoresIndividuos(self, populacao) -> tuple: ···
12
13      def reproduzir(self, populacao, qtdeCidades, cidadeInicial) -> None: ···
39
```

Resultados do sistema

Conforme mencionado na subseção Codificação, o sistema gera dois tipos de saídas para o utilizador, uma saída no terminal do sistema operacional e uma interface gráfica com apresentação das cidades geradas, conforme explicado na subseção citada.

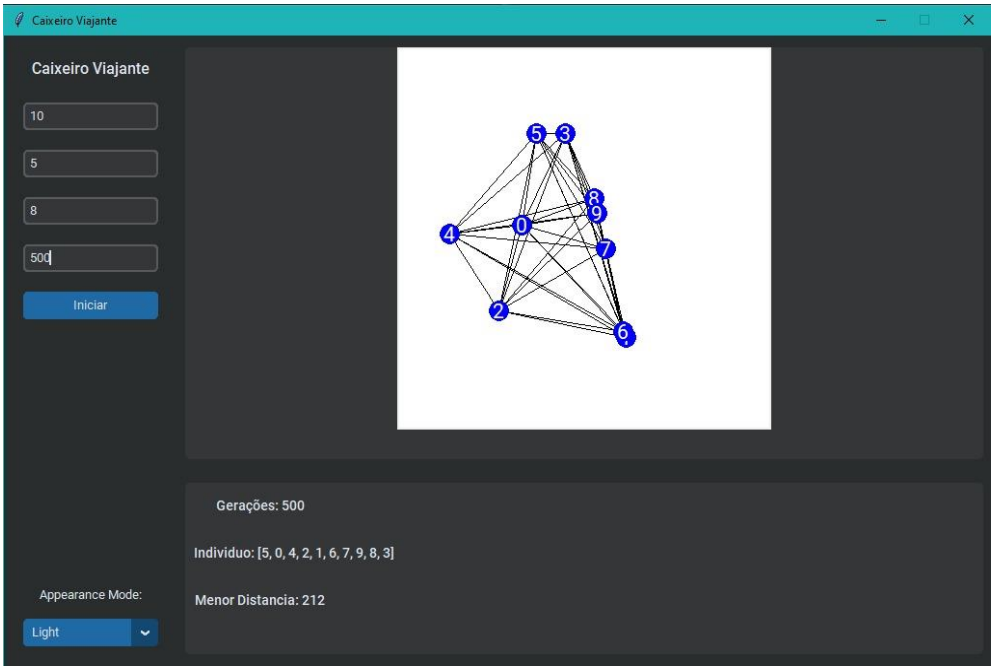
Conforme citado, a saída por interface do terminal é completa, como representa a imagem abaixo, informando os dados de cada geração do sistema, apresentando muitas linhas em tela e no final o resultado. Para evitar dados desnecessários, recortou-se um trecho do resultado.

Figura 2. Interface do terminal do sistema operacional

```
Populacao:
[[3, 5, 6, 8, 2, 1, 4, 9, 7, 0], [3, 5, 6, 8, 2, 1, 4, 9, 7, 0], [3, 5, 6, 8, 2, 1, 4, 9, 7, 0], [3, 5, 6, 8, 2, 1, 4, 9, 7, 0], [3, 5, 6, 8, 2, 1, 4, 9, 7, 0]
Fitness:
[284, 284, 284, 284, 284, 284]
Geracao: 99
-=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=-
Populacao:
[[3, 5, 6, 8, 2, 1, 4, 9, 7, 0], [3, 5, 6, 8, 2, 1, 4, 9, 7, 0], [3, 5, 6, 8, 2, 1, 4, 9, 7, 0], [3, 5, 6, 8, 2, 1, 4, 9, 7, 0], [3, 5, 6, 8, 2, 1, 4, 9, 7, 0]
Fitness:
[284, 284, 284, 284, 284, 284]
Geracao: 100
-=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=--=-
Solução encontrada:
Individuo:  [3, 5, 6, 8, 2, 1, 4, 9, 7, 0]
Menor distancia:  284
```

A interface gráfica apresenta uma forma diferente para analisar o sistema, com desenho das cidades geradas, podendo observar as linhas geradas para os cálculos de distância, no bloco abaixo do desenho observa-se o resultado dos cálculos de forma simples e objetiva

Figura 3. Interface do sistema codificado



CONSIDERAÇÕES FINAIS

Os objetivos propostos neste estudo foram completamente atingidos. O objetivo principal, quando iniciamos o projeto, era de finalizarmos com o desenvolvimento de um sistema para calcular a rota mínima no setor de logística, e conclui-se que o problema do caixeiro viajante torna-se recorrente no âmbito da área de logística, podendo ser um agravante, principalmente, em custos para ela, quando adicionado o uso do algoritmo genético para criação de uma inteligência artificial constata-se que é suficiente para atingir o objetivo de criação de um sistema para auxílio dessas empresas, entende-se que seja um programa funcional para o ramo alvo do projeto, pois possibilita efetuar em um curto espaço de tempo muitos cálculos de rota, redução de custos em entregas e demoras desnecessárias para o produto chegar ao cliente final.

REFERÊNCIAS BIBLIOGRÁFICAS

BRITO, José André de M; MONTENEGRO, Flávio Marcelo Tavares. Um algoritmo genético para o problema de agrupamento. XXXVIII Simpósio brasileiro pesquisa operacional, 2006. Disponível em: <http://din.uem.br/~ademir/sbpo/sbpo2006/pdf/arq0196.pdf>. Acesso em: 14 dez. 2022.

CARVALHO, André Ponce de Leon F. de. Redes neurais artificiais. Ciência da Computação: Universidade de São Paulo, 2000. Disponível em: <https://sites.icmc.usp.br/andre /research/neural/>. Acesso em: 17 out. 2022.

CASTRO, Higor Tomaz Teixeira de; MOREIRA, Ildeu de Castro; MASSARANI, Luiza. Percepções da teoria da evolução e seleção natural em comentários no youtube. Interfaces Científicas - Humanas e Sociais, [S. l.], v. 9, n. 3, p. 184–201, 2022. Disponível em: <https://periodicos.set.edu.br/humanas/article/view/10723>. Acesso em: 13 out. 2022.

LOPÉZ, Romão Del Cura. Custo logístico na distribuição ao varejo. Revista de estudos acadêmicos – FAE Business. Curitiba: UniFAE, p. 39-41, jun. 2005.

PACHECO, Marcos Aurélio Cavalcanti. Algoritmos genéticos: princípios e aplicações. Pontifícia Universidade Católica do Rio de Janeiro. PUC-RIO: Rio de janeiro, 1999. Disponível em: < http://www.inf.ufsc.br/~mauro.roisenberg/ine5377/Cursos-ICA/CE-intro_apost .pdf >. Acesso em: 20 out. 2022.

ARARIPE, Raynner Braga; KLOECKNER, Natália Varela da Rocha. Problema do caixeiro viajante (pcv) aplicado a otimização de roteiros de veículos de transporte rodoviário de uma distribuidora de óleo lubrificante a granel em fortaleza e região metropolitana. Revista de Engenharia da UNI7, Fortaleza, p 1 -49, 2017. Disponível em: < https://periodicos.uni7.edu.br/index.php/revista-de-engenharia/article/download/526/326/>. Acesso em: 28 nov. 2022.

SANTOS, Charles Morphy Dias; SILVA, Mariane Tavares. Uma análise histórica sobre a seleção natural: de darwin-wallace à síntese estendida da evolução. Amazônia: Revista de Educação em Ciências e Matemáticas, Belém, v. 11, n. 22, p. 46-61, 2015. Disponível em: <https://periodicos .ufpa.br/index.php/revistaamazonia/ article/view/2122>. Acesso em: 03 nov. 2022.

SILVA, Lucas Natali Magalhães. Tipos de aprendizado de máquina e algumas aplicações. Terra Lab: Universidade Federal de Ouro Preto. 2021. Disponível em: < http://www2.decom.ufop.br/terra lab/tipos-de-aprendizado-de-maquina-e-algumas-aplicacoes/>. Acesso em: 20 nov. 2022.

SILVEIRA, J. F. Porto da. Problema do caixeiro viajante. Complexidade computacional do problema do caixeiro. Matemática elementar: Universidade Federal do Rio Grande do Sul. 2000. Disponível em: <http://www.mat.ufrgs.br/~portosil/caixeiro.html>. Acesso em: 21 nov. 2022.

TAULLI, Tom. Introdução a inteligência artificial. Uma abordagem não técnica. São Paulo. Apress Media. 2020.